Euler Tours on the Grid

Maria Astefanoaei

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2014

Abstract

The problem of finding whether an undirected graph has an Euler Tour or not has a very simple polynomial-time algorithm. However, there are no efficient known algorithms for exactly or approximately counting the number of Euler Tours on undirected graphs, except in some special cases (for example on graphs of bounded treewidth [5]). In this project we are trying to experimentally study the Markov chain using "Kotzig moves" on Euler Tours.

Acknowledgements

I am very thankful to my supervisor, Dr. Mary Cryan, for her continuous guidance, thorough feedback and genuine enthusiasm for the project.

Table of Contents

1 Introduction **1**

2 Background **3**
2.1 Basic definitions . . . 3
2.2 Counting Euler Tours . . . 4
2.2.1 Decision and counting problems . . . 4
2.2.2 Complexity of counting . . . 5
2.3 Markov Chains . . . 7
2.3.1 Conductance . . . 9
2.3.2 Canonical paths . . . 9
2.4 The Kotzig Chain . . . 10
2.4.1 Grids and Transition Systems . . . 10
2.4.2 Kotzig moves on the grid . . . 11
2.4.3 The Chain . . . 12

3 Sampling Euler Tours of low height grids **14**
3.1 The transfer matrix approach . . . 14
3.1.1 Inductive approach . . . 15
3.1.2 Computing the transfer matrix . . . 17
3.2 The Number of tours of the $(2 \times n)$ and $(3 \times n)$ Grids . . . 18
3.2.1 $(2 \times n)$ Grid . . . 18
3.2.2 $(3 \times n)$ Grid . . . 21
3.3 Generating Euler Tours . . . 26
3.3.1 Sampling Knapsack solutions . . . 26
3.3.2 Dynamic programming approach for sampling Euler Tours . . . 27
3.4 Illustrative example . . . 29

4 Experimental analysis of the mixing time of the Kotzig Chain **35**
4.1 Conductance and Canonical Paths . . . 35
4.2 Building the paths - Theory . . . 36
4.3 Building the paths - The Algorithm . . . 39
4.4 Illustrative example . . . 41
4.5 Results . . . 43

5 Conclusions **45**

Bibliography **46**

Chapter 1

Introduction

The problem of exactly or approximately counting the number of Euler Tours on undirected graphs proved to be difficult to tackle, except for some very specific cases, for example graphs of bounded treewidth [5]. In this dissertation we studied the mixing time of a Markov chain that uses a special kind of transformations on a particular graph structure. The goal was to experimentally evaluate whether the chain may be rapidly mixing, which would indicate whether there might be an efficient approximation algorithm for the number of Euler Tours on the grid.

The inspiration for the approach came from an earlier attempt of Tetali and Vemapala [18] to prove the "Kozig moves" chain is rapidly mixing on all regular graphs of degree 4. However, it is well-known that their proof does not hold [10], so we considered an experimental approach. This approach was feasible because there exists an exact counting algorithm for Euler Tours on toroidal grids, which allowed us to uniformly generate uniform random tours on these graphs. To give a rough idea, the project followed the following steps: first we sampled a big set of Euler Tours of the grid, which we grouped into pairs; for each of these pairs we have built paths from one tour to the other, using the *Kotzig*-moves (or κ-transformations). We were hoping to demonstrate using these kind of paths that the Kotzig Chain is likely to be rapidly mixing.

The contributions of the project are:

1. reading and understanding the literature related to the $\sharp$P-complexity class, running time of Markov Chains, conductance, canonical paths, sampling and approximation algorithms, and Kotzig transformations on grids
2. reading, understanding, filling in the gaps and correcting the transfer matrix algorithm of Creed [3]
3. adapting Dyer's [6] dynamic programming algorithm to count Euler Tours of small grids
4. writing a sampling algorithm and generating a collection of pairs of Euler Tours based on the above table

5. building paths between tours using a rule inspired by Tetali and Vempala's attempt to prove the Kotzig Chain is rapidly mixing
6. getting an estimate of the edge loading.

To be able to present the experimental settings and motivate our approach we began by giving basic definitions related to graphs, followed by a discussion on the complexity of counting problems; we then continued Chapter 2 with a general description of Markov chains and their properties, also presenting the canonical paths technique we have used hoping to bound the conductance of the Kotzig chain, in order to demonstrate it is likely to be rapidly mixing. After this general presentation we gave details about the particular graph structures we have worked with and the kind of transformations needed to build the Kotzig chain. In Chapter 3 we present the exact counting algorithm for Euler Tours of low-height grids [3], correct some errors from [3] and show how to implement exact counting and sampling via a dynamic programming table. In Chapter 4 we described how we have built the paths, backed up by the existing literature. We end both chapters by working through examples in order to illustrate the steps involved in the described algorithms. In Chapter 5 we present our conclusions.

Chapter 2

Background

In this chapter we describe general notions and techniques related to Euler Tours and Markov Chains and their mixing time. In the first section we describe the problem of finding Euler Tours on a graph, which is followed by a presentation of the counting problem in the second section. After arguing the difficulty of counting Euler Tours and proving the need to find an approximation algorithm, we described how Markov Chains work and what kind of properties they need to have to be rapidly mixing, a necessary condition for developing efficient sampling/approximately counting algorithms. We continue Section 2.3 by presenting, in general, the "canonical paths" technique with which we hoped to show the chain we are working with has good behaviour. We end Chapter 2 with a detailed description of the chain we are interested in, the Kotzig chain, with its space states and transitions explained.

2.1 Basic definitions

We will begin by giving a few standard definitions and results related to graphs and Euler Tours.

Definition 1. An *undirected graph* $G = (V, E)$ is a set of vertices V and a set of edges E such that each $e \in E$ is an unordered pair of vertices, $e = \{u, v\}$, where $u, v \in V$. The *degree* of a vertex $v \in V$ is the number of edges $e \in E$ containing v and we denote it by *deg(v)*.

Definition 2. Let $G = (V, E)$ be a graph. An *Euler Tour* of G is a cycle that uses every edge exactly once. Two Euler Tours are equivalent if one is a cyclic permutation of the other. We denote the set of all Euler Tours of a graph G by $ET(G)$.

Definition 3. An undirected graph $G = (V, E)$ is said to be *Eulerian* if it is connected and all vertices have even degree.

The following result was proved by Euler in the 18^{th} century:

Euler's Theorem. *An undirected graph $G = (V,E)$ contains an Euler Tour if and only if G is a connected graph and every vertex has even degree.*

Proof. Suppose G contains an Euler Tour; then it is necessarily connected. For every edge "entering" a vertex there needs to be exactly one "exiting". This gives an even degree for every vertex.

The other implication will be proved by induction on the number of edges.

Base case: For a graph with one vertex and no edges, there is one possible tour.

Induction step: Suppose any graph with the two properties and having less than n edges is Eulerian. Given a graph $G = (V,E)$ with n edges, choose an arbitrary vertex $v \in V$ and starting from v build a random walk that only visits each edge once. Continue the walk until there are no more unvisited edges at the vertex most recently visited. Since all the vertices have even degree, there are either 0, 2 or 4 unvisited edges incident to each visited vertex, except for vertex v, which will have only 1 unvisited edge. Therefore the walk will necessarily stop at v. After removing the visited set of edges from the graph we will be left with a finite set of connected components all with vertices of even degree. According to the induction step all these components are Eulerian. Moreover, all of them intersect the initial cycle in at least one vertex (because G is a connected graph). Therefore we exchange the pairings of some of the vertices to build a longer tour. In the end we will have used all vertices and visited all edges.

□

To check whether a graph has an Euler Tour is therefore not too difficult - it can be solved in linear time. We only need to check that every vertex has even degree and also do a breadth first search to verify the graph is connected. A more challenging problem is to count the number of Euler Tours in the case of Eulerian graphs, which will be discussed in the next section.

2.2 Counting Euler Tours

Once it is known that a graph has at least one Euler Tour, a natural question to be asked is how many such tours can be found. It was proved that exactly counting the number of Euler Tours is a difficult problem, belonging to the ♯P-complete complexity class [2]. In the next few sections general notions of complexity theory will be discussed, in order to illustrate what it means for a problem to be ♯P-complete and to motivate the need to find a way to approximate the number of Euler Tours of a graph as opposed to exactly counting them.

2.2.1 Decision and counting problems

In the 1970's, when computational complexity started being studied more intensively, scientists were mostly concerned with decision problems. Intuitively, a decision prob-

lem in computational complexity is one asking for the existence of a solution, for example "is there a path between two certain vertices in a graph" or "is this graph Eulerian?". Valiant [19], one of the most renowned scientists in the field, pointed out that the counting versions of the problems they studied were sometimes much more interesting, which led to him defining a new complexity class for these type of problems. For each decision problem there is an associated counting problem, which asks for the *number of solutions* as opposed to asking whether a solution exists. For the earlier examples the associated counting problems are:"how many different paths between two certain vertices in a graph are there?" or "how many different Euler Tours does a given graph have?" No matter how easy the decision problem is, its counting equivalent does not necessarily have a known efficient algorithm. There are exceptions, such as the problem of counting the spanning trees of an undirected graph with no self-loops; an efficient algorithm for this problem takes $O(n^3)$ time in the number of vertices [13]. However, the majority of interesting counting problems, including the one we are concerned with, are not known to have polynomial-time algorithms. We call these counting problems $\sharp\mathsf{P}$-complete, which we will formally define in the next section.

2.2.2 Complexity of counting

First, before defining the $\sharp\mathsf{P}$ class, we will define NP, the class of decision problems which can be verified in polynomial time. Formally:

Definition 4. Let Σ be a finite alphabet and $R \subset \Sigma^* \times \Sigma^*$. For a particular instance $x \in \Sigma^*$, the decision problem R asks whether there exists some $y \in \Sigma^*$ such that $(x,y) \in R$. We say a decision problem R is in the class NP (non-deterministic polynomial time) if

1. there exists a polynomial $p(\cdot)$ such that for any instance $x \in \Sigma^*$ we have
$$|y| \leq p(|x|) \ \forall y \in \Sigma^* \text{ such that } (x,y) \in R;$$
2. there exists a polynomial time algorithm for testing the predicate $(x,y) \in R$.

Similarly, the $\sharp\mathsf{P}$ class is defined as:

Definition 5. Let Σ be a finite alphabet. Given a relation $R \subset \Sigma^* \times \Sigma^*$ we can define the counting problem for R as the problem of computing a function $N_R : \Sigma^* \to \mathbb{N}$ with values
$$N_R(x) = |\{y \in \Sigma^* : (x,y) \in R\}|.$$
We say the counting problem N_R is in the class $\sharp\mathsf{P}$ if

1. there exists a polynomial $p(\cdot)$ such that for any instance $x \in \Sigma^*$ we have
$$|y| \leq p(|x|) \ \forall y \in \Sigma^* \text{ such that } (x,y) \in R;$$
2. there exists a polynomial time algorithm for testing the predicate $(x,y) \in R$.

We will also define what it means for a computational problem to be hard or complete, but first we need to introduce the notion of reduction.

Definition 6. For a set of functions F from $\mathbb{N}$ to $\mathbb{N}$ and two sets S_1 and S_2 we say that S_1 is F-reducible to S_2 if $\exists f \in F, \forall x \in \mathbb{N}$ such that:

$$x \in S_1 \iff f(x) \in S_2.$$

If R is a computation problem such that any NP problem can be reduced to R via a polynomial-time computable function, then R is called NP-*hard*; if R is itself NP, it is NP-*complete*. Similarly, if any ♯P problem can be reduced to R via a polynomial-time computable function, then R is called ♯P-*hard*; if R is also in the ♯P class itself, it is ♯P-*complete*.

As was noted before, some easy decision problems may have ♯P-complete counting equivalents. The first such problem that was studied, counting the perfect matchings of a graph, was proved to be ♯P-complete[19], even though the decision problem is in P. The time complexity of finding one perfect matching is $O(|VE|)$, where V is the set of vertices and E the set of edges. The same goes for the problem of counting Euler Tours - even though finding one tour is $O(|E|)$, to count all tours is a ♯P-complete problem.[2]

A decision problem can be treated as a subcase of a counting problem, so NP is included in ♯P and it is believed that ♯P-complete problems are even more computationally difficult than NP-complete ones. We do not expect to find a polynomial time algorithm for a ♯P-complete problem, since that would imply that P = NP. However, Jerrum, Valiant and Vazirani [12] have discovered that for any ♯P self-reducible problem we can either find a polynomial approximation algorithm or the problem cannot be approximated at all in polynomial time. Specifically, we are looking for polynomial time algorithms that, using randomness, produce an estimate that is sufficiently close to the real answer. This is formally defined as:

Definition 7. [12] Let Σ be a finite alphabet and let $f : \Sigma^* \to \mathbb{N}$ be a counting problem on Σ. A randomised approximation scheme for f with confidence parameter δ is a randomised algorithm that takes as input an instance $x \in \Sigma^*$ and an error parameter ε, and outputs a number $N \in \mathbb{N}$ (this is a random variable of the "coin tosses" made by the algorithm),

$$\mathbb{P}[(1-\varepsilon)f(x) \leq N \leq (1+\varepsilon)f(x)] \geq 1-\delta.$$

We call this a fully polynomial randomised approximation scheme, or a FPRAS, if the algorithm runs in time bounded by a polynomial in $|x|, \varepsilon^{-1}$ and $log(\delta^{-1})$, for every instance x.

Jerrum, Valiant and Vazirani [12] have found that for self-reducible relations building an approximate counter FPRAS is equivalent to building an approximate sampler FPAUS. There are obvious differences between counting and sampling, but using two algorithms the two tasks become connected.

2.3 Markov Chains

In many areas, such as statistical physics and combinatorial optimization, the Markov Chain Monte Carlo method has proved to be really useful in finding a FPRAS for computationally difficult problems when other approaches have failed. For example, the monomer-dimer problem, a classical problem from physics, could only be tackled using this approach[11]. Formally, a Markov Chain is defined as:

Definition 8. A discrete-time Markov chain M with finite state space Ω is a stochastic process $(X_t)_{t\geq 0}$, with $X_t \in \Omega$ for all $t = 1, 2, ...$, that satisfies the Markov property:

$$P[X_{t+1} = y | X_t = x_t, X_{t-1} = x_{t-1} ... X_0 = x_0] = P[X_{t+1} = y | X_t = x_t].$$

We can define a Markov chain by its transition probability matrix P:

$$P(x,y) = \mathbb{P}[X_{t+1} = y | X_t = x] \ \ \forall x, y, \in \Omega.$$

More commonly, especially when $|\Omega|$ is exponential in its description we give a "rule" for making transitions instead of a matrix.

Now we describe the Markov Chain Monte Carlo technique. The goal is to try to estimate the cardinality of a very large given set of combinatorial structures, call it Ω. The Markov Chain helps sample at random an element of this set according to a uniform probability distribution π on Ω. If we choose the chain to be *ergodic*, meaning that the distribution over Ω asymptotically converges to π, and if we simulate the chain for enough steps, we are sure that the sample will be from a probability distribution sufficiently close to π. In order for the algorithm to be efficient, a small number of steps should be needed (where by small we mean polynomial in the size of the input); if this is the case we say that the Markov chain is *rapidly mixing*. Therefore, three properties of the Markov chain are required to find a reliable approximation: having the correct stationary distribution, ergodicity, and rapid mixing time, all of which are defined below:

Definition 9. Let $\mathcal{M}$ be a time-homogeneous Markov chain with state space Ω and transition probability matrix P. A distribution $\pi : \Omega \to [0,1]$ is called a *stationary distribution* of $\mathcal{M}$ if

$$\sum_{x \in \Omega} \pi(x) P(x,y) = \pi(y) \ \ \forall y \in \Omega.$$

Definition 10. Let $\mathcal{M}$ be a time-homogeneous Markov chain with state space Ω and transition probability matrix P. $\mathcal{M}$ is said to be *ergodic* if it is aperiodic and irreducible, where those concepts are defined as:

$\mathcal{M}$ is aperiodic if $\forall x \in \Omega$, $\gcd\{t : P^t(x,x) > 0\} = 1$;

$\mathcal{M}$ is irreducible if$\forall x, y \in \Omega$, $\exists t$ such that $P^t(x,y) > 0$.

To sample from the stationary distribution of $\mathcal{M}$ we need to run the chain for an infinite number of steps. Since this is impossible, we will have to choose a sufficiently large

number of steps such that we know the distribution is "close enough" to the stationary distribution. Formally, to know for how long we need to run the chain before it converges, we need to compute the "mixing time" of the chain.

Definition 11. Let $\mathcal{M}$ be a finite, discrete time Markov chain with state space Ω. For $\varepsilon > 0$, the *mixing time* of $\mathcal{M}$ from initial state $x \in \Omega$,$\tau_x(\varepsilon)$, is defined as

$$\tau_x(\varepsilon) = min\{t : ||P^t(x,\cdot) - \pi||_{TV} \leq \varepsilon\}.$$

We define the *mixing time* of M , $\tau(\varepsilon)$, to be the maximum over the mixing times from each state:

$$\tau(\varepsilon) = \sup_{x\in\Omega} \tau_x(\varepsilon).$$

Definition 12. A Markov chain $\mathcal{M}$ on state space Ω is said to be *rapidly mixing* if, for all initial states $x \in \omega$ and all $\varepsilon > 0, \tau_x(\varepsilon)$ is bounded above by some function which is polynomial in $|x|$. A Markov chain is said to be *torpidly mixing* if there exist some $\varepsilon > 0$ and $x \in \omega$ for which $\tau_x(\varepsilon)$ is bounded below by some function exponential in $|x|$.

To prove how Markov Chain Monte Carlo simulation works, we have chosen to illustrate an example described by Jerrum and Sinclair [11], namely the knapsack problem. We will use the same problem later to illustrate how to sample an object of the state space uniformly at random.

Suppose we are given a vector $a = (a_1, a_2, \ldots, a_n)$, where $a_i \in \mathbb{N}$ and a value $b \in \mathbb{N}$. The counting problem is to find the number of vectors $x = (x_1, x_2, \ldots, x_n)$, where each x_i can be either 0 or 1, such that $a \cdot x \leq b$. The state space Ω is the set of solutions, that is the set of vectors x which make the above equality correct.

There exists a naive Monte Carlo simulation for this problem, but it is very inefficient. Instead, we will consider the following approach: suppose we are at some state x of the chain, a solution of the form $x = (x_1, x_2, \ldots, x_n)$; we will either remain there with probability $\frac{1}{2}$ or otherwise, with probability $\frac{1}{2}$ we select an x_i uniformly at random and go to the state $x = (x_1, x_2, \ldots, (1-x_i), \ldots, x_n)$ if it is a solution to the knapsack problem (otherwise remain in the same state).

The Markov Chain defined this way is ergodic and the stationary distribution is uniformly distributed over Ω. We can find a FPRAS that approximates the number of solutions by expressing $|\Omega|$ as a product of smaller factors, and approximating each of them with Monte Carlo simulations [11]. We should note though that this method runs in polynomial time only if we can prove that the Markov Chain associated is "rapidly mixed".

To decide if the Markov Chain we are interested in is "rapidly mixing", we study its *conductance* - informally, a large conductance means there is no "blockage" in the chain, which means it has a fast mixing time. Because calculating this quantity directly involves the whole state space Ω and considering all its subsets, we will instead use a technique called "canonical paths" to study the conductance of the chain. We will formally define these notions below, after which we will present the specific Markov Chain of Euler Tours we studied, describing the combinatorial structures involved and how the transitions between states are performed.

2.3.1 Conductance

As mentioned earlier, we wish to experimentally study the conductance of the Markov Chain in order to gain insight in whether it may be rapidly mixing. Let $\mathcal{M} = (\Omega, P)$ be a discrete state, time-homogeneous, Markov chain with stationary distribution π, where Ω is the state space of the chain.

Definition 13. [11] For any non-empty subset $S \subset \Omega$ the *conductance* $\Phi(S)$ of S is defined to be

$$\Phi(S) = \frac{\sum_{x \in S, y \in \Omega \setminus S} \pi(x) P(x,y)}{\pi(S)}.$$

Then the *conductance* of $\mathcal{M}$ is the minimum over all sets:

$$\Phi(M) = \min_{\substack{S \subset \Omega \\ 0 < \pi(S) \leq 1/2}} \Phi(S).$$

For a particular subset S of the state space, we can think of $\Phi(S)$ as the "jumping out" probability corresponding to that set - a low value would be an indicator of a blockage in the chain. To show this, Jerrum and Sinclair have proved the following theorem:

Theorem 1. *[11] Let $M = (\Omega, P)$ be a time-homogeneous, reversible Markov chain and suppose $P(x,x) \geq 1/2$ for all $x \in \Omega$. Then, the mixing time, $\tau(\varepsilon)$, and conductance, $\Phi(\mathcal{M})$, satisfy*

1. *For all $x \in \Omega$, $\tau_x(\varepsilon) \leq \frac{2}{\Phi(\mathcal{M})}(\log(\varepsilon^{-1} + \log(\pi(x)^{-1})))$;*
2. *$max_{x \in \Omega} \tau_x(\varepsilon) \geq \frac{1-2\Omega(\mathcal{M})}{e\Omega(\mathcal{M})} log(\varepsilon^{-2})$.*

Therefore, to prove that our chain is rapidly mixing we would need to show that $1/\Phi$ is bounded above by a polynomial in n. The usual way to prove bounds on conductance is via the "canonical paths" technique, which requires paths with certain global properties to be defined for every pair in $\Omega \times \Omega$. This option was explored by Tetali and Vempala [18], but there were found some errors in their proof [10]. We modified their approach hoping to get insight into whether the Markov Chain may be rapidly mixing.

2.3.2 Canonical paths

A path γ_{xy} is described in [11] as a sequence of transitions that takes us from a start state x of the Markov Chain $\mathfrak{M}$ to a final state y. In the "canonical paths method" we define a canonical path γ_{xy} for every pair in $\Omega \times \Omega$. Given this definition, let $\Gamma = \{\gamma_{xy}\}$ be the set of paths from any x to any y and let E denote the set of edges of the state space. To prove the rapid mixing time we would need to choose a collection Γ that achieves a good loading. We measure the loading by:

$$\bar{\rho} = \bar{\rho}(\Gamma) = \max_e \frac{1}{Q(e)} \sum_{\gamma_{xy} \ni e} \pi(x)\pi(y)|\gamma_{xy}|,$$

where γ_{xy} denotes the path from x to y, $|\gamma_{xy}|$ is the length of the path, $Q(x,y) = \pi(x) \cdot P(x \to y) = \pi(y) \cdot P(y \to x)$ and the maximum is over the edges of the state space graph.

In order for a Markov chain to be rapidly mixing, it needs to have no "bottlenecks" - there should be a set of paths Γ such that $\rho(\Gamma)$ is not too big. One thing that is believed to lead to a good edge loading is choosing the paths in a "canonical" way. Sinclair formalized this in the following way:

Proposition. *[17] Let $\mathfrak{M}$ be a finite, reversible, ergodic Markov chain with loop probabilities $P(x,x) \geq \frac{1}{2}$ for all states x. Let Γ be a set of canonical paths with maximum edge loading $\bar{\rho} = \bar{\rho}(\Gamma)$. Then the mixing time of $\mathfrak{M}$ satisfies*

$$\tau_x(\varepsilon) \leq \bar{\rho}(ln\pi(x)^{-1} + ln\varepsilon^{-1}),$$

for any choice of initial state x.

Jerrum and Sinclair [11] give an example of the usage of the canonical paths method by proving that the Markov Chain induced by a random walk on the hypercube is rapidly mixing. However, in general, finding a collection of good paths is particularly hard - $\bar{\rho}$ depends on the size of the state space, which is the quantity we were trying to estimate in the first place. To overcome this, the canonical paths method usually uses a quasi-injective map that avoids explicitly counting the elements of the state space.

We have used the canonical paths technique to study the mixing time of the Markov Chain with the state space Euler Tours of grids (the combinatorial structures we will discuss next), whose transitions are local transformations at vertices, named "Kotzig moves". Throughout the rest of the report we will refer to this chain as the "Kotzig Chain". In our approach we followed:

- First we sampled a subset of Euler Tours for the $(2 \times n)$ and $(3 \times n)$ grids with small n, from which we picked pairs at random.
- Then we have built paths in an ordered fashion using Kotzig moves for each pair of tours.
- We counted the number of times each edge of the state space graph that was used in some path.

In the next two sections we will give details about the specific structures we have built the paths on and how each transition was realised.

2.4 The Kotzig Chain

2.4.1 Grids and Transition Systems

The type of graphs we will be working with is the toroidal grid, defined below.

Definition 14. The $m \times n$ toroidal grid, denoted $G(m,n)$, is the 4-regular graph with vertex set $\{(i,j) : 0 \leq i < m, 0 \leq j < n\}$ and edges joining each (i,j) to $(i, j \pm 1 \bmod n)$ and to $(i \pm 1 \bmod m, j)$.

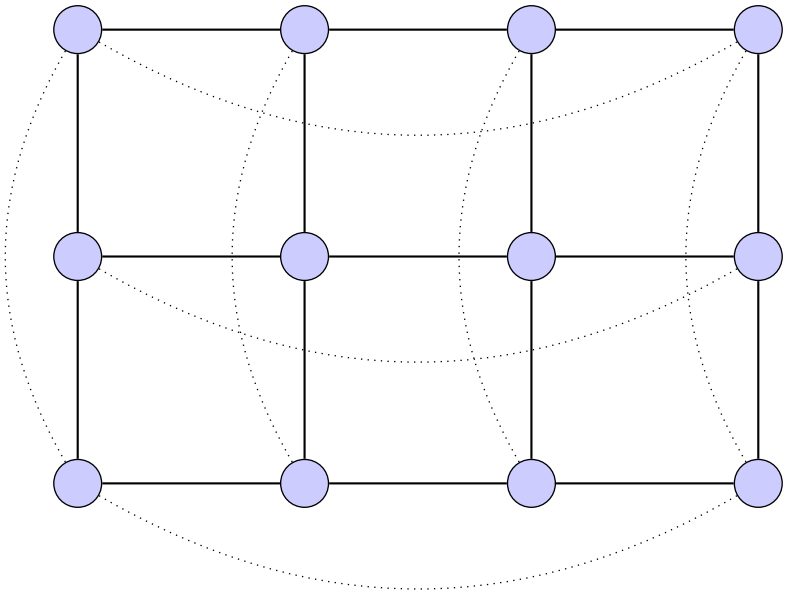


Figure 2.1: $G(2,4)$

A very convenient way to represent Euler Tours, suggested in [1] and [14], is to enumerate each pairing of edges visited at each vertex - the pair being formed of the "entry" edge and the "exit" edge that are visited consecutively in the tour.

Definition 15. [18] Let $V = \{v_1, ..., v_n\}$ be the set of vertices of $G = (V,E)$. Then a *transition system* T of G is defined as $T = \{T(v_1), ..., T(v_n)\}$, where $T(v_i)$ is an arbitrary pairing of the edges incident at v_i. By a *pairing* of edges we mean a partitioning of edges incident at a vertex into (unordered) pairs of edges. By a *transition* we mean a pair of edges in the transition system.

Each transition system T splits the grid into a set of cycles; if there is only one cycle, then that is the Euler Tour defined by T. It is not always the case that there is one single cycle in the decomposition, therefore not all transition systems yield an Euler Tour.

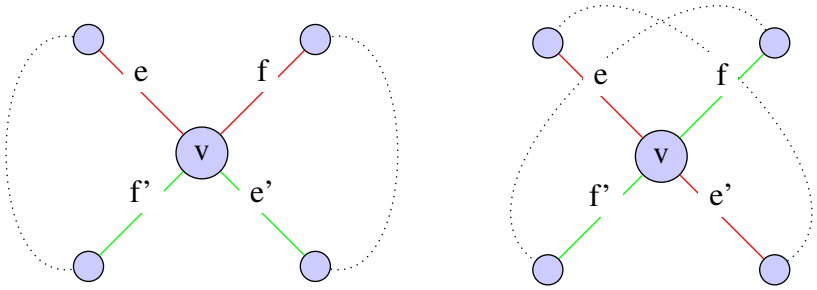
2.4.2 Kotzig moves on the grid

In [1], Kotzig and Abrham have shown that in order to transform one Euler Tour of a graph into a different one, all is needed is a finite set of simple local transformations - called κ-transformations.

Definition 16. Let $G = (V,E)$ be an Eulerian graph and let $\mathcal{T}$ be an Euler Tour of G. Say the sequence of edges visited by the tour at a vertex v is $...e(v)f...e'(v)f'....$ Then

a κ-transformation will change this sequence to $...e(v)e'...f(v)f'...$ which corresponds to a different Euler Tour of the graph G.

Graphically, this could be represented as:



κ-transformation

Kotzig and Abrham [1] introduce the notions of *allowed* and *prohibited* transitions, that are also used by Tetali and Vempala [18]:

Definition 17. Given two transition systems T,T', we use $T \Rightarrow_v T'$ to denote that we can transform $T(v)$ into $T'(v)$ by a κ-transformation at v and call this an *allowed* transition; to denote the contrary, we use $T \not\Rightarrow_v T'$, and call it a *prohibited* transition.

On a 4-regular graph, for each of the 3 possible pairings at a vertex, there is only one possible κ-transformation and it corresponds to an allowed transition; the other pairing corresponds to a prohibited transition.

2.4.3 The Chain

We can now describe the Markov chain defined by Tetali and Vempala [18]. The state space is the set of Euler Tours of a graph and the transitions are κ-transformations. More precisely, the steps are:

1. Choose an arbitrary Euler Tour of the graph $G = (V,E)$.
2. From the set of vertices V pick one uniformly at random.
3. With probability $1/2$ make the only possible κ-transformation at v and with the remaining probability do nothing.

The chain is ergodic and it converges to the uniform distribution of the set of Euler Tours [3], but it has not been proved to be rapidly mixing. As Jerrum observed[10], Tetali and Vempala's attempt to prove that the Kotzig Chain is rapidly mixing has failed because they have assumed the paths have a series of global properties which do not actually hold. Researchers in Markov Chains have attempted to fix their proof but without success, because of the complexity of the structures involved. More details on their approach and why it did not work will be given in Section 4.2. We are hoping

that by choosing a different way to build the paths (which will be presented in Section 4) we will achieve a good edge loading that would suggest the Kotzig Chain may be rapidly mixing.

Chapter 3

Sampling Euler Tours of low height grids

In order to study the Kotzig Chain we need to have a big set of Euler Tours from which we can pick the pairs of tours that will give us the paths. In general, we do not have an algorithm to uniformly sample Euler Tours. However, in the case of low height toroidal grids we can exactly count the number of Euler Tours as shown by Creed for the $2 \times n$ and $3 \times n$ grids [**?**].

To sample Euler Tours we have used a dynamic programming approach similar to Dyer's algorithm for sampling knapsack solutions [6]. The first stage is to build a table iteratively, at each step adding the number of paths corresponding to some cycle decomposition of tours. Golin et al. [7] argue that, by using the transfer matrix approach (which will be presented in the first section), one could find the number of certain combinatorial structures on some types of graphs. Creed [3] has used this to show how to compute in theory the number of tours of toroidal grids and applied it to small $(2 \times n)$ and $(3 \times n)$ grids. We have also used the transfer matrix technique to build the dynamic programming table for sampling $(2 \times n)$ and $(3 \times n)$ grids. In the first section of the chapter we gave details of the approach and described how Creed has found the exact form of the matrix in order to compute the number of Euler Tours of grids. Because his presentation for the $(2 \times n)$ and $(3 \times n)$ grids is succinct and missing quite a few steps, we have decided to include in the second section all the details of the matrix computation, also referring to our implementation. After that, we described how the dynamic programming table is built and how the sampling works, and in the last section we show all the steps of the approach with the help of an example.

3.1 The transfer matrix approach

Golin et al. [7] made the observation that in grids, cylinders and toruses with fixed height, a collection of graph structures (e.g., spanning trees, Hamiltonian cycles, independent sets, acyclic orientations) should be partitionable in terms of the configuration on the left and right columns. Moreover, the counts of such structures can probably

be built up inductively as columns are added. For an appropriate transfer matrix A and corresponding row vectors x and y the computation should be achievable as

$$|ET(G(m,n))| = \vec{x}A^n\vec{y}^T.$$

Creed [3] has taken further steps in this direction by defining and showing how the transfer matrix for Euler Tours on the toroidal grid can be computed in the general case and providing the exact expressions for the $(2 \times n)$ and $(3 \times n)$ grids. In the following sections we will explain how the inductive construction is done and how it is used to compute the number of tours. After that we will describe how to find the appropriate transfer matrix based on Creed's work.

3.1.1 Inductive approach

Before showing why the above formula works we will first define a few useful notions.

Definition 18. A *transition system* T of an Eulerian graph $G = (V,E)$ at a vertex v is a decomposition of the set of edges incident with v into pairs. A transition system of G is a function T that maps each v to a transition system at v.

Each transition system T splits the grid into a set of cycles; if there is only one cycle, then that is the Euler Tour defined by T. Obviously it is not always the case that there is one single cycle in the decomposition, therefore not all transition systems yield an Euler Tour.

Definition 19. A *partial transition system* is a transition system defined on

$$V_k = \{(i,j) : 0 \leq i \leq m-1, 0 \leq j \leq k-1\}, V_0 = \emptyset.$$

If such a partial transition system can be extended to a transition system on G that defines an Euler Tour, then it is a *legal partial transition system*.

Definition 20. Let $\mathcal{P}(m)$ be the set of perfect matchings on

$$\{l_i, r_{i,k} | 0 \leq i \leq m-1, 1 \leq k \leq n\},$$

where $l_i = \{(i,0),(i,n-1)\}, r_{i,k} = \{(i,k-1),(i,k)\}$, for each $0 \leq i \leq m-1$ and a fixed $1 \leq k \leq n$.

A more correct notation would be $\mathcal{P}(m,k)$, where we also parametrize with respect to k, but considering that all $\mathcal{P}(m,k)$ are isomorphic for a fixed m and all k, a simpler notation was chosen.

Definition 21. To each legal partial transition system T on G we assign a *class* $C \in \mathcal{P}(m)$, where the edges in C correspond to the endpoints of the paths in the decomposition of V_k induced by T.

The transfer matrix A is defined such that each cell $A(C,C')$, with $C,C' \in \mathcal{P}(m)$ will contain the number of legal partial transition systems on $V_{k+1}\backslash V_k = \{(i,k) : 0 \leq i \leq m-1\}$ which extend a transition system on V_k with class C to a transition system on V_{k+1} with class C'. Then A is a $P(2m) \times P(2m)$ matrix, where $P(2m)$ is the number of perfect matchings on K_{2m}, the complete graph with $2m$ vertices. It is easy to see that the number of transition systems doesn't depend on k.

In a similar fashion with the structure of Golin's [7] transfer matrix, A is block diagonalizable "with very special blocks" (many of which are 0-blocks in the $2 \times n$ and $3 \times n$ cases), which "reduces the size of its characteristic polynomial", making computation less difficult.

It is useful to consider V_k because it allows a recursive construction of the grid (as in Golin et. al [7]) - we can extend $G(m,n)$ to $G(m,N)$ (with m fixed, $N > n$) by separating the first and last columns and keeping the edges attached only to the first column, adding columns at the end of the graph until we reach length N and reattaching the connecting edges between the first column and the new last column. This also gives intuition into why the transfer matrix A needs to be raised to the n^{th} power in the expression for the total number of Euler Tours. If the cell $A(C,C')$ contains the number of transition systems on $V_{k+1}\backslash V_k$ (which connects a configuration with class C on k columns to a configuration with class C' on $k+1$ columns), then the corresponding cell in A^s will represent the number of transition systems on $V_{k+s}\backslash V_k$, where s is any positive integer. To prove this inductively consider the $p = P(2m)$ classes of transition systems: $C_1, C_2, ..., C_p$. $A(C_i,C_j)$ will give the number of transition systems from C_i to C_j for only one column; when adding another column we need to consider all possible combinations that can get us from C_i to C_j: transform C_i to C_w and C_w to C_j for $1 \leq m \leq p$ which can be done in $A(C_i,C_w) \times A(C_w,C_j)$ ways. If we choose w to be each number from 1 to p, we get that the number of transitions on the two columns is:

$$\sum_{w=1}^{p} A(C_i,C_w) \cdot A(C_w,C_j),$$

which is just the product between row i and column j from A. Doing the same for all i's and j's we obtain that the transfer matrix for two columns is just A^2. Obviously, the proof is the same if we add a column to k columns, where k is any positive integer less than n. Therefore, A^n will give the number of transition systems on the subgraph induced by $V_n\backslash V_0$, which looks exactly like $G(m,n)$ after we merge the edges attached exclusively to the first column with the ones attached only to the last column.

Now come into play the row vectors x and y from the expression of $|ET(G(m,n))| = \vec{x}A^n\vec{y}^T$. To make sure we connect the right edges, we only consider the row in A corresponding to the class of transitions that uses all the horizontal edges; this means that $x(C) = 1$ only for that class C that contains $l_i, r_{i,n-1}$ for all $0 \leq i \leq m-1$. This assures that all rows in $G(m,n)$ will get connected again. Moreover, we only need to consider those classes that will make a complete cycle, so we set $y(C) = 1$ only if identifying l_i with r_i will give rise to a cycle.

With A,x,y defined in this way, we now need to show how to find the number of

transitions for each pair of classes.

3.1.2 Computing the transfer matrix

To explain how the values in the transfer matrix are computed, we will start by looking at an example. Consider the $G(2,n)$ grid with its 3 possible classes of transition systems: $[l_0r_0]$, $[l_0r_1]$, $[l_0l_1]$. Say the transition system on V_k (the first k columns) has class $[l_0, r_1]$:

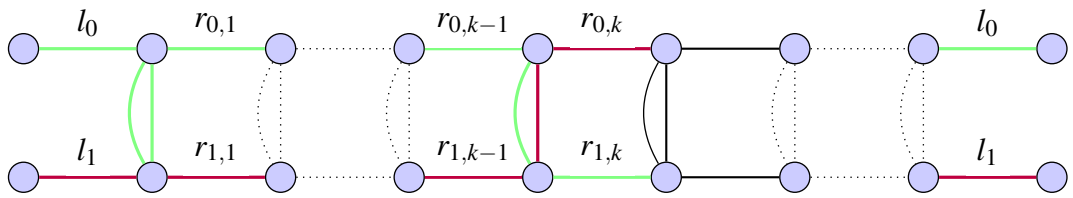
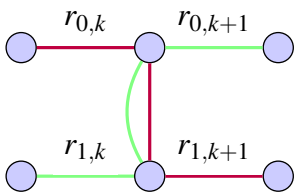


Figure 3.1: $TS(V_k)$ with class $[l_0, r_1]$

and we wanted the partial transition system on V_{k+1} to have class $[l_0, r_0]$. One pairing of the edges that would satisfy this is, for example:



Identifying the corresponding edges we now have that the transition system of V_{k+1} is $[l_0, r_0]$:

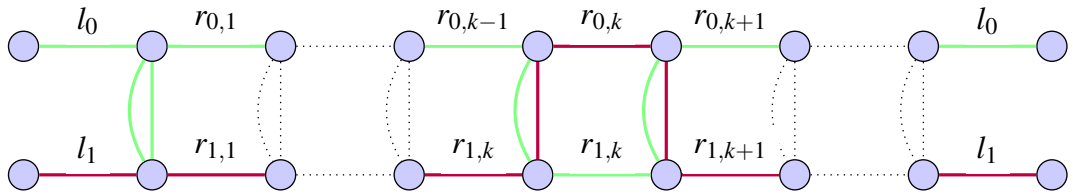


Figure 3.2: $TS(V_{k+1})$ with class l_0, r_0

In general, Creed observes that the only illegal transition system (ie. not extendable to an Euler Tour of the whole grid) on $V_{k+1} \setminus V_k$ is the one which gives the full cycle formed by the vertical edges on column k [3]. He then proceeds to classify the two possible types of pairings between edges, which helps define the set of matchings corresponding to transition systems on a particular column. Whenever we add the pairings of the vertices of a column, we identify the edges on the right of the vertices

of the previous column with the edges on the left of the vertices on the current column. This process will either give us a cycle or a set of two paths; since having a cycle means we will not be able to build an Euler Tour, we must only count those pairings that give two paths and also form the right transition system.

By choosing an arbitrary matching on $\{r_{i,k}, r_{i,k+1} | 0 \leq i \leq m-1\}$ for some k we need to consider the transition systems that yield this particular matching. Note that we can decompose the cycle $(0,1,...,m-1)$ into either a set of paths with disjoint edges or a single cycle. In each case the number of transition systems will be a sum of any of the elements in the set $\{0,1,2,2m\}$.

3.2 The Number of tours of the $(2 \times n)$ and $(3 \times n)$ Grids

Creed [3] gives the transfer matrices and the corresponding row vectors x and y for the $(2 \times n)$ and $(3 \times n)$ grids, but with no details of how those values were found. This section aims to fill in those gaps, which will be helpful for explaining the sampling algorithm and its implementation.

3.2.1 $(2 \times n)$ Grid

In the $2 \times n$ case the only classes are: $[l_0, r_{0,k}]$, $[l_0, r_{1,k}]$, $[l_0, l_1]$ (3 classes, which corresponds to the number of perfect matchings on the complete graph with 6 vertices).

The matrix A will therefore be a 3×3 matrix:

	$[l_0, r_0]$	$[l_0, r_1]$	$[l_0, l_1]$
$[l_0, r_0]$	4	2	2
$[l_0, r_1]$	2	4	2
$[l_0, l_1]$	0	0	6

If we would like to work out, for example, the value $A([l_0, r_0], [l_0, r_0])$, we need to consider the following transition system:

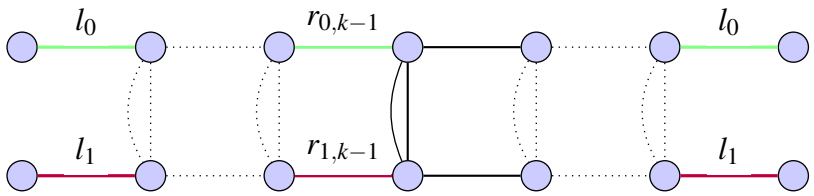


Figure 3.3: Class $[l_0, r_0]$

and think of how we can achieve this:

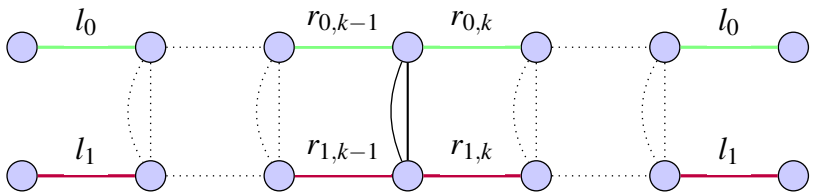
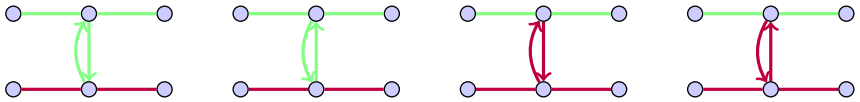


Figure 3.4: Class $[l_0, r_0]$

There are 4 possible ways to do this - the edges corresponding to the vertices on the k-th column can be paired in any of the following ways:

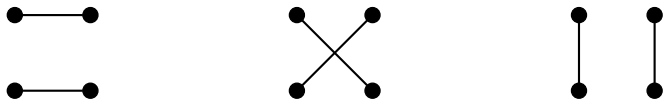


where by arrows we indicate which vertical edge pairs with which horizontal edge.

The value in any $A(C,C')$ cell is the number of different pairings induced by one or more classes of transition systems on a column of the grid. With this in mind, we have built a *transition table*, in which we take a note of the classes of partial transition systems that allow the transformation from C to C'. This transition table is:

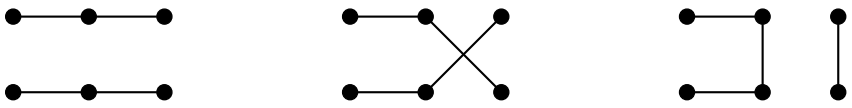
	$[l_0,r_0]$	$[l_0,r_1]$	$[l_0,l_1]$
$[l_0,r_0]$	$[l_0,r_0]$	$[l_0,r_1]$	$[l_0,l_1]$
$[l_0,r_1]$	$[l_0,r_1]$	$[l_0,r_0]$	$[l_0,l_1]$
$[l_0,l_1]$	–	–	$[l_0,r_0]$ and $[l_0,r_1]$

To see why this works, we will represent, inspired by Creed's notation, each of the classes by:



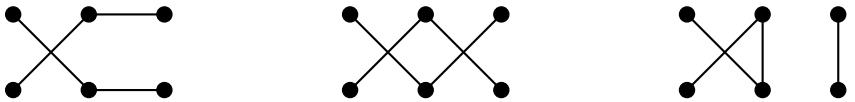
$[l_0,r_0]$ $[l_0,r_1]$ $[l_0,l_1]$

Then, starting with $[l_0,r_0]$, we can obtain the following classes:



$[l_0,r_0]+[l_0,r_0] \to [l_0,r_0]$ $[l_0,r_0]+[l_0,r_1] \to [l_0,r_1]$ $[l_0,r_0]+[l_0,l_1] \to [l_0,l_1]$

Similarly, starting with $[l_0,r_1]$:



$[l_0,r_1]+[l_0,r_0]\to[l_0,r_1]$ $[l_0,r_1]+[l_0,r_1]\to[l_0,r_0]$ $[l_0,r_1]+[l_0,l_1]\to[l_0,l_1]$

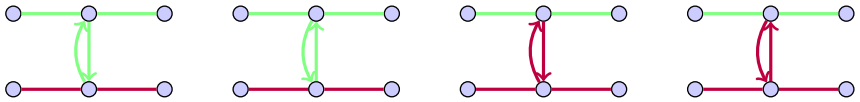
If the transition class of a partial transition system is $[l_0,l_1]$, the class of the transition system of any of the following subgrids will also be $[l_0,l_1]$, so any of the other classes cannot subsequently be reached.

$[l_0,l_1]+[l_0,r_0]\to[l_0,l_1]$ $[l_0,l_1]+[l_0,r_1]\to[l_0,l_1]$

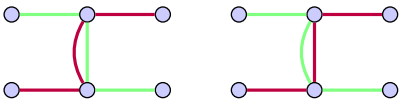
(So $[l_0,r_1]+[l_0,r_0]\to[l_0,r_1]$ means that in the transition table $([l_0,r_1],[l_0,r_1])=[l_0,r_0]$.)

From these, the above transition table follows immediately. Now we need to find the actual pairings for all the classes. We will then substitute the class in the transition table with the number of pairings corresponding to that class in order to get the transfer matrix.

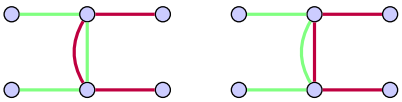
For l_0,r_0 the 4 corresponding partial transition systems are:



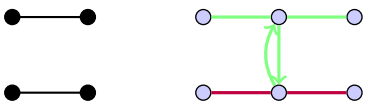
For $[l_0,r_1]$ the 2 corresponding transition systems are:



For l_0,l_1 the 2 corresponding transition systems are:



Note that the two graphs:



mean different things. The first graph represents a class of transition systems and the other is one partial transition system corresponding to that class.

We then obtain A by filling in the corresponding number of pairings given by the transition table.

	$[l_0,r_0]$	$[l_0,r_1]$	$[l_0,l_1]$
$[l_0,r_0]$	$[l_0,r_0]$	$[l_0,r_1]$	$[l_0,l_1]$
$[l_0,r_1]$	$[l_0,r_1]$	$[l_0,r_0]$	$[l_0,l_1]$
$[l_0,l_1]$	$-$	$-$	$[l_0,r_0]$ and $[l_0,r_1]$

$\rightarrow$

	$[l_0,r_0]$	$[l_0,r_1]$	$[l_0,l_1]$
$[l_0,r_0]$	4	2	2
$[l_0,r_1]$	2	4	2
$[l_0,l_1]$	0	0	6

To find the vector x for the $G(2,n)$ grid, remember that $x(C) = 1$ if $[l_i,r_i] \in C$ for all i and $x(C) = 0$ otherwise. In this case the transition classes are $[l_0,r_0],[l_0,r_1],[l_0,l_1]$, therefore only $x([l_0,r_0]) = 1$, which gives $x = (1,0,0)$. For y, $y(C) = 1$ if identifying l_i with r_i in C gives rise to a single m-cycle and $y(C) = 0$ otherwise. Hence $y([l_0,r_1]) = 1$ and $y([l_0,l_1]) = 1$, which gives $y = (0,1,1)$. Carrying out the calculations we find out that the number of tours of the $(2 \times n)$ grid is $(2n+3)6^{n-1} - 2^{n-1}$.

3.2.2 $(3 \times n)$ Grid

In a similar way we can find the number of Euler Tours of the $3 \times n$ grid. Considering the number of transition system of all the 15 possible classes (corresponding to the 15 perfect matching of the K_6 complete graph), Creed [3] builds the transfer matrix, that, as Golin et al. [7] suggested it should, has a "specific structure" [8], e.g. block diagonalizable with very special blocks. The matrix given by Creed is:

$$\begin{pmatrix} A_1 & B_1 & C_1 & C_1 & C_1 \\ B_1 & A_1 & C_1 & C_1 & C_1 \\ 0 & 0 & D_1 & 0 & 0 \\ 0 & 0 & 0 & D_1 & 0 \\ 0 & 0 & 0 & 0 & D_1 \end{pmatrix}$$

where $A_1 = 5I_3 + J_3$, $B_1 = 2J_3$, $C_1 = I_3 + J_3$ and $D_1 = 2I_3 + 6J_3$ (I_3 is the 3×3 identity matrix and J_3 is the 3×3 matrix in which all entries are 1).

However, while implementing the sampling algorithm, we realized that some of the entries in this matrix should be different. The top right 6×9 submatrix (corresponding to the 6 blocks of C_1) should be instead:

$$\begin{array}{ccccccccc} 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 & 2 & 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 2 \\ 1 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \end{array}$$

and also $D_1 = 3I_3 + 5J_3$. The values in D_1 are not very concerning because D_1 is still symmetrical and we know it would not "break" A's block structure, but we could not be sure if the same would happen for the top right 6×9 matrix.

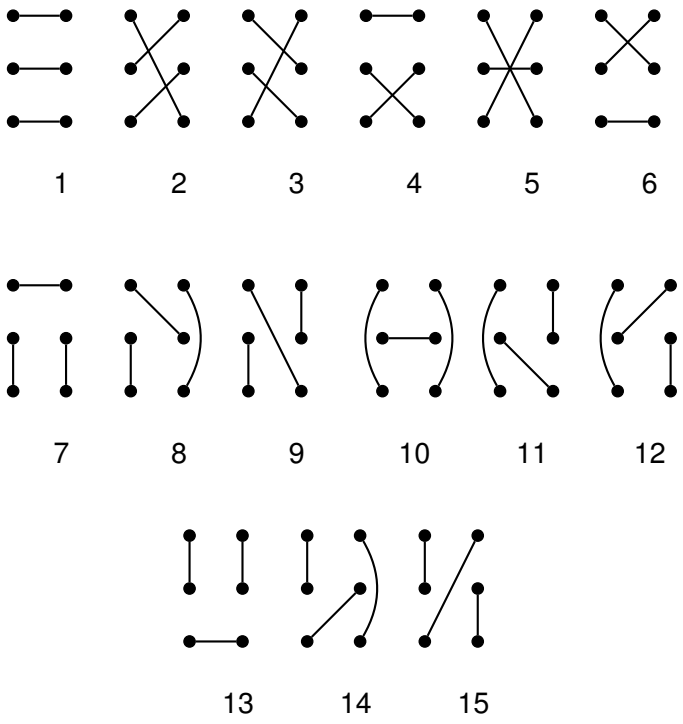
We initially assumed that changing the order of the classes in this matrix would eventually lead to the nice form presented by Creed. It turns out that no matter how we would shuffle the rows and columns, obtaining that form is impossible, because the two submatrices have different ranks. However, the block structure is preserved when raising the matrix to a power, which is what we are concerned with.

Let $C_1 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$, $C_1^* = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{pmatrix}$, $C_1^{**} = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}$ and A_s be the matrix with the shuffled order:

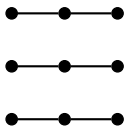
$$A_s = \begin{pmatrix} A_1 & B_1 & C_1^* & C_1^* & C_1 \\ B_1 & A_1 & C_1^* & C_1^{**} & C_1^* \\ 0 & 0 & D_1 & 0 & 0 \\ 0 & 0 & 0 & D_1 & 0 \\ 0 & 0 & 0 & 0 & D_1 \end{pmatrix}$$

We will prove at the end of this section that we can still use the transfer matrix method, even though the matrix does not have the typical structure. For now, we will use the new matrix and show how some of the entries were computed.

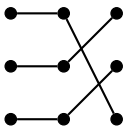
There are 15 perfect matchings on K_6, the complete graph with 6 vertices, therefore there are 15 transition classes for the $(3,n)$ grid:



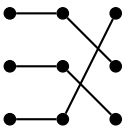
As before, we search for all the classes that can make the transition from one class to another. We will give more details for the A_1 matrix, which looks at classes [1],[2] and [3]:



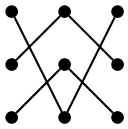
$[1]+[1]\rightarrow[1]$



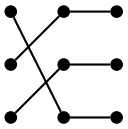
$[1]+[2]\rightarrow[2]$



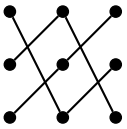
$[1]+[3]\rightarrow[3]$



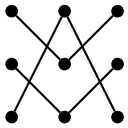
$[2]+[3]\rightarrow[1]$



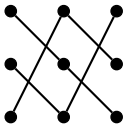
$[2]+[1]\rightarrow[2]$



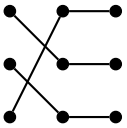
$[2]+[2]\rightarrow[3]$



$[3]+[2]\rightarrow[1]$



$[3]+[3]\rightarrow[2]$



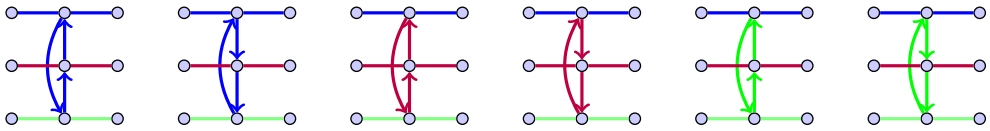
$[3]+[1]\rightarrow[3]$

This gives us the transition table for A_1:

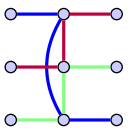
Class	1	2	3
1	1	2	3
2	3	1	2
3	2	3	1

After this, we had to find all the possible transition systems corresponding to each transition class.

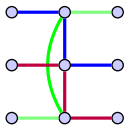
Class [1] - 6 transition systems:



Class [2] - 1 transition system:



Class [3] - 1 transition system:



Substituting the appropriate values in the above transition table we get A_1, as given by Creed:

$$A_1 = \begin{pmatrix} 6 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 6 \end{pmatrix}.$$

We went through the same steps for all the other submatrices of the transfer matrix A. The associated transition table is:

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	1	2	5	6	4	15	14	13	8	9	7	11	10	12
3	2	3	1	6	4	5	12	10	11	14	13	15	9	8	7
4	4	5	6	1	2	3	7	8	9	14	13	15	11	10	12
5	5	6	4	3	1	2	15	14	13	10	11	12	9	8	7
6	6	4	5	2	3	1	12	10	11	8	9	7	13	14	15
7	–	–	–	–	–	–	4,1	3,6	2,5	–	–	–	–	–	–
8	–	–	–	–	–	–	2,6	5,1	3,4	–	–	–	–	–	–
9	–	–	–	–	–	–	3,5	2,4	6,1	–	–	–	–	–	–
10	–	–	–	–	–	–	–	–	–	5,1	3,4	2,6	–	–	–
11	–	–	–	–	–	–	–	–	–	2,4	6,1	3,5	–	–	–
12	–	–	–	–	–	–	–	–	–	3,6	2,5	4,1	–	–	–
13	–	–	–	–	–	–	–	–	–	–	–	–	6,1	2,4	3,5
14	–	–	–	–	–	–	–	–	–	–	–	–	3,4	5,1	2,6
15	–	–	–	–	–	–	–	–	–	–	–	–	2,5	3,6	4,1

and substituting the classes in this table with the number of transition systems in each corresponding class (or pair of classes when there is more than one) we obtain the modified version of the transfer matrix.

3.2.2.1 The Shuffled Transfer Matrix

We will prove by induction that A^n and A_s^n have the same values, but in the shuffled order. Remember that A and A_s have the following forms:

$$A = \begin{pmatrix} A_1 & B_1 & C_1 & C_1 & C_1 \\ B_1 & A_1 & C_1 & C_1 & C_1 \\ 0 & 0 & D_1 & 0 & 0 \\ 0 & 0 & 0 & D_1 & 0 \\ 0 & 0 & 0 & 0 & D_1 \end{pmatrix} \qquad A_s = \begin{pmatrix} A_1 & B_1 & C_1^* & C_1^* & C_1 \\ B_1 & A_1 & C_1^* & C_1^{**} & C_1^* \\ 0 & 0 & D_1 & 0 & 0 \\ 0 & 0 & 0 & D_1 & 0 \\ 0 & 0 & 0 & 0 & D_1 \end{pmatrix}$$

It is not too difficult to see that A^n is:

$$A^n = \begin{pmatrix} A_n & B_n & C_n & C_n & C_n \\ B_n & A_n & C_n & C_n & C_n \\ 0 & 0 & D_n & 0 & 0 \\ 0 & 0 & 0 & D_n & 0 \\ 0 & 0 & 0 & 0 & D_n \end{pmatrix}$$

where $\begin{pmatrix} A_1 & B_1 \\ B_1 & A_1 \end{pmatrix}^n = \begin{pmatrix} A_n & B_n \\ B_n & A_n \end{pmatrix}$, $D_1^n = D_n$ and $C_n = A_1C_{n-1} + B_1C_{n-1} + D_{n-1}C_1$.

Suppose that A_s^n is:

$$A_s^n = \begin{pmatrix} A_n & B_n & C_n^* & C_n^* & C_n \\ B_n & A_n & C_n^* & C_n^{**} & C_n^* \\ 0 & 0 & D_n & 0 & 0 \\ 0 & 0 & 0 & D_n & 0 \\ 0 & 0 & 0 & 0 & D_n \end{pmatrix},$$

where C_n^* and C_n^{**} have the same entries as C_n but in the order given by C_1^* and C_1^{**} respectively. For example, if $C_n = \begin{pmatrix} c'_n & c_n & c_n \\ c_n & c'_n & c_n \\ c_n & c_n & c'_n \end{pmatrix}$ then $C_n^* = \begin{pmatrix} c'_n & c_n & c_n \\ c_n & c_n & c'_n \\ c_n & c'_n & c_n \end{pmatrix}$.

Clearly, the left 6×6 submatrix in A_s^{n+1} will be:

$$\begin{pmatrix} A_1 & B_1 \\ B_1 & A_1 \end{pmatrix}^{n+1} = \begin{pmatrix} A_{n+1} & B_{n+1} \\ B_{n+1} & A_{n+1} \end{pmatrix}$$

and the bottom right 9×9 one:

$$\begin{pmatrix} D_1 & 0 & 0 \\ 0 & D_1 & 0 \\ 0 & 0 & D_1 \end{pmatrix}^{n+1} = \begin{pmatrix} D_{n+1} & 0 & 0 \\ 0 & D_{n+1} & 0 \\ 0 & 0 & D_{n+1} \end{pmatrix},$$

so the same as in A^{n+1}.

The entries in the 6×9 top right submatrix will be given by:

$$\begin{pmatrix} A_1C_n^* + B_1C_n^* + D_nC_1^* & A_1C_n^* + B_1C_n^{**} + D_nC_1^* & A_1C_n + B_1C_n^* + C_1D_n \\ B_1C_n^* + A_1C_n^* + C_1^*D_n & B_1C_n^* + A_1C_n^{**} + D_nC_1^{**} & B_1C_n + A_1C_n^* + D_nC_1^* \end{pmatrix}$$

Because $B_1 = 2J_3$, then $B_1C_n = B_1C_n^* = B_1C_n^{**}$. We also have that $A_1C_n + B_1C_n + C_1D_n = C_{n+1}$, so we can conclude that:

$$\begin{array}{rcl} A_1C_n + B_1C_n^* + C_1D_n & = & C_{n+1} \\ A_1C_n^* + B_1C_n^* + D_nC_1^* & = & A_1C_n^* + B_1C_n^{**} + D_nC_1^* \quad = \\ B_1C_n^* + A_1C_n^* + C_1^*D_n & = & B_1C_n + A_1C_n^* + D_nC_1^* \end{array}$$

We can also see that $B_1C_n^* + A_1C_n^{**} + D_nC_1^{**} = C_{n+1}^{**}$ and $B_1C_n + A_1C_n^* + D_nC_1^* = C_{n+1}^*$ by either carrying out the calculations or by observing that, as before $B_1C_n = B_1C_n^*$ and also A_1C_n has the same entries as $A_1C_n^{**}$ in the order of C^{**} and $D_nC_1^{**}$ has the same entries as D_nC_1, again, in the order of C^{**} (given that A_1 and D_1 are of the form $xI_3 + yJ_3$, they both preserve the order). Similarly for the second equality. Therefore,

$$A_s^{n+1} = \begin{pmatrix} A_{n+1} & B_{n+1} & C_{n+1}^* & C_{n+1}^* & C_{n+1} \\ B_{n+1} & A_{n+1} & C_{n+1}^* & C_{n+1}^{**} & C_{n+1}^* \\ 0 & 0 & D_{n+1} & 0 & 0 \\ 0 & 0 & 0 & D_{n+1} & 0 \\ 0 & 0 & 0 & 0 & D_{n+1} \end{pmatrix},$$

which proves that the block structure is preserved, A_s^{n+1} having the same entries as A^{n+1} but in the fixed different order.

3.3 Generating Euler Tours

In order to sample Euler Tours we have used a technique similar to the one Dyer used to sample solutions for a variant of the knapsack problem with the purpose of approximating the total number of solutions[6]. The algorithm is based on building a dynamic programming table.

3.3.1 Sampling Knapsack solutions

To briefly remind the reader, the knapsack problem is: suppose we are given a vector $a = (a_1, a_2, \ldots, a_n)$, where $a_i \in \mathbb{N}$ and a value $b \in \mathbb{N}$. We wish to find the number of vectors $x = (x_1, x_2, \ldots, x_n)$, where each x_i can be either 0 or 1, such that $a \cdot x \leq b$. For the purposes of this section we will only present Dyer's approach to sample uniformly one solution of a given instance of the problem.

We denote by S the set of solutions from which we would like to sample uniformly. Starting from the initial knapsack problem, Dyer considers a different but very similar problem for which we can easily (in polynomial time) compute the number of solutions. We have that

$$\sum_{j=1}^{n} a_j \cdot x_j \leq b,$$

which we can rewrite as

$$\sum_{j=1}^{n} \frac{a_j n^2}{b} \cdot x_j \leq n^2.$$

Let $\alpha_j = \left\lfloor \frac{a_j n^2}{b} \right\rfloor$, and let S' be the set solutions of

$$\sum_{j=1}^{n} \alpha_j \cdot x_j \leq n^2,$$

which has a very similar form to the initial knapsack problem. To build the dynamic programming table that gives the number of solutions to this problem, Dyer considered the number of solutions of all its smaller variants:

$$F(r,s) = |\{x| \sum_{j=1}^{r} \alpha_j \cdot x_j \leq s^2\}|.$$

He then observed that the following recurrence relation holds:

$$F(r,s) = F(r-1,s) + F(r-1,s-\alpha_r),$$

with the initial condition

$$F(1,s) = \begin{cases} 1, \text{ if } s < \alpha_1 \\ 2, \text{ otherwise.} \end{cases}$$

Using dynamic programming we can tabulate a table with these values in $O(n^3)$ time, and noting that $F(n,n^2) = |S'|$, we have therefore found the number of solutions to the very similar problem considered. Dyer also proves that $1 \leq |S|/|S'| \leq (n+1)$, which means we can use $|S'|/\sqrt{n+1}$ to approximate $|S|$.

The way we can find a uniform solution S' is by starting with the (n,n^2) cell of the table and tracing back probabilistically. Knowing that $F(n,n^2) = F(n-1,n^2) + F(n-1,n^2-\alpha_n)$, we can consider $\frac{F(n-1,n^2)}{F(n,n^2)}$ the probability of x_n to be 1 and with the remaining probability $\frac{F(n-1,n^2-\alpha_n)}{F(n,n^2)}$ we choose x_n to be 0. Depending on which choice we have made, we either go to the cell $F(n-1,n^2)$ or to $F(n-1,n^2-\alpha_n)$, where we repeat the process. In this way we will find a vector $x = (x_1,x_2,\ldots,x_n)$ which is in S' and with probability $1/(n+1)$ it is also in S. If x is not in S we start again from cell $F(n,n^2)$.

3.3.2 Dynamic programming approach for sampling Euler Tours

We will now describe how we have built the dynamic programming table that we are using as support to sample Euler Tours.

Consider a (m,n) grid and let $p = P(2m)$ where $P(2m)$ is the number of perfect matchings on K_{2m}, the complete graph with $2m$ vertices. Remember from the previous section that for grids the following relation holds:

$$|ET(G(m,n))| = \vec{x}A^n\vec{y}^T,$$

where A is the $p \times p$ transfer matrix for that grid and has size and x and y are two row vectors of size $1 \times p$.

The dynamic programming table has size $p \times (n+2)$, and, similarly to A, x and y each row corresponds to one of the p transition classes of the grid (see Section 3.1 for more details about transition classes). We will consider the indexing on the table starts from 1 and not 0. We initialized the table with the values in x - the first column is exactly x. The next column will contain the values of $\vec{x}\cdot A$, the following one will be $\vec{x}\cdot A^2$ and so on until the $(n+1)$-th column, which will be $\vec{x}\cdot A^n$. The last column contains only the value $\vec{x}\cdot A^n\vec{y}^T$, so the total number of tours. Each of the values in this table corresponds to the count of certain path cycle decompositions that can create an Euler Tour. To be more precise, the value on row r and column $(c+1)$ represents the number of paths corresponding to a tour in which the transition system on the first c columns of the grid has class r. At the end we are multiplying by y to make sure we only choose those paths that can give an Euler Tour.

The recurrence relations used to build the table are:

$$T(r,c) = \begin{cases} x(r) & c = 1 \\ \sum_{i=1}^{p} T(i,c-1)\cdot A(i,r) & c > 1, c < n+2 \\ 0 & c = n+2, r < p \\ \sum_{i=1}^{p} T(i,n+1)\cdot y(i) & c = n+2, r = p \end{cases}$$

so for all columns from 2 to $n+1$ in the table $T(r,c)$ is the dot product between the transpose of the previous column in the table and the r-th column in A, while the other 2 columns are x and a column containing only zeros and the number of tours.

Now that we have built the table, we can trace back probabilistically in the same way in which Dyer did to find a solution for the knapsack problem.

We obtain an Euler Tour in the following way: start with the bottom right cell, namely $T(p,n+2)$, whose value, by the above formula, is $\sum_{i=1}^{p} T(i,n+1)\cdot y(i)$. With each step back we choose a cell on the previous column; as we know, the row of the cell corresponds to a transition class. Therefore, with probability $T(i,n+1)\cdot y(i)/T(p,n+2)$ we choose class i for the transition system of the first n columns of the graph. Suppose we move to cell $T(i,n+1)$, where with probability $T(j,c-1)\cdot A(j,i)/T(i,n+1)$ we choose transition class j for the transition system of the first $n-1$ columns of the grid. We continue to do this until we reach the second column of our table, at which point we stop. We now have n transition classes, with the i-th class being the class of the transition system of the first i columns.

1	X		X			0
0		X				0
0				X	X	$\lvert ET(G(2,5))\rvert$
	$c_1 = C_1$	$c_2 = C_2$	$c_3 = C_1$	$c_4 = C_3$	$c_5 = C_3$	

Backtracking on the dynamic programming table
Starting with $T(3,7)$ - the bottom right cell - we went to cell $T(3,6)$ corresponding to class C_3, then to $T(3,5)$, so class C_3, then to $T(1,4)$, corresponding to class C_1 and so on.

Say the chosen classes are: $c_1, c_2, ..., c_n$. We need to find the actual pairings of the edges that give these classes. For each $i \leq n$ we choose a pairing of the edges of the vertices on column i that makes the transition from class c_{i-1} to c_i (we consider c_0 to be the class that uses all the horizontal edges). The number of possible pairings for a column i is exactly the value $A(c_{i-1}, c_i)$, where A is the transfer matrix. With probability $1/A(c_{i-1}, c_i)$ we choose one of these pairings. We continue to do this until we reach the class of the last column.

Once we have the pairings for all the vertices of the grid, we need to put them together to form the tour.

3.4 Illustrative example

To show the steps of the sampling algorithm we will work through an example. In general, we represent $2 \times n$ grids as:

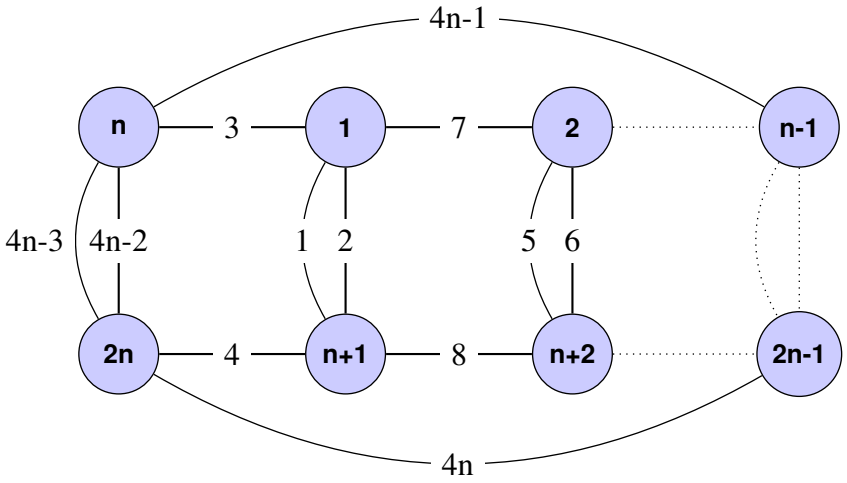


Figure 3.5: $G(2,n)$

Observe that a column k is represented by:

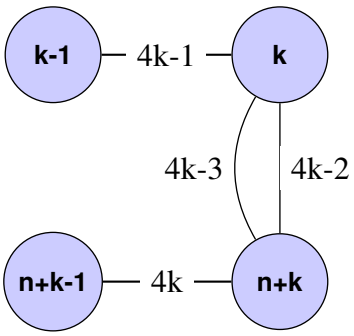


Figure 3.6: $G(2,n)$

In code we store the possible pairings of edges as an array where each row corresponds to a class of transition systems; we enumerate all the possible pairings corresponding to every class. So for example, for class $[l_0, r_0]$, the pairings are:

```
pairings[0] = new int[][][][]
          {{{{ 3, 1 }, { 2, 7 }}, {{ 4, 8 }, { 1, 2 }}},
           {{{ 3, 2 }, { 1, 7 }}, {{ 4, 8 }, { 1, 2 }}},
           {{{ 3, 7 }, { 2, 1 }}, {{ 4, 2 }, { 1, 8 }}},
           {{{ 3, 7 }, { 1, 2 }}, {{ 4, 1 }, { 2, 8 }}}};
```

where the first set is the pairing of edges for vertex 1 and the second set is the pairing for vertex $n+1$. This of course corresponds to the edges on the first column. When we need to get the pairings for column k we update these values to match the edges of the vertices on column k. Say we have chosen `pairing [0][0]` for column 4; then we will update $\{\{3,1\},\{2,7\}\},\{\{4,8\},\{1,2\}\}$ to $\{\{15,13\},\{14,19\}\},\{\{16,20\},\{13,14\}\}$ because 3 corresponds to edge $4k-1$, 1 is in the same position as $4k-3$ and so on (if the grid has exactly 4 columns we need to take each value mod 16).

Suppose we are trying to sample an Euler Tour for the $(2,4)$ grid, which in code is represented as:

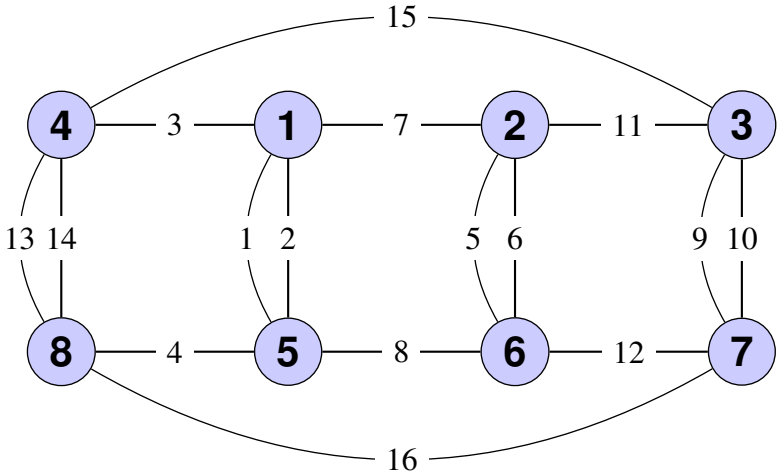


Figure 3.7: $G(2,4)$

We first need to build the dynamic programming table. Remember that the transfer matrix, x and y vectors are:

$$A = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 4 & 2 \\ 0 & 0 & 6 \end{pmatrix}, x = (1,0,0), y = (0,1,1).$$

Each row in the table corresponds to a class of transition systems, the first column is the initialization and the last one gives us the total number of tours (so the 4 in between correspond to the 4 columns in the grid for which we need to find a transition system to build the tour). We initialize the table with the x vector:

1					
0					
0					

We multiply this vector by matrix A to get the next column:

1	4				
0	2				
0	2				

We multiply again the last added vector with A:

1	4	20			
0	2	16			
0	2	24			

And so on, until we reach the 5-th column of the table:

1	4	20	112	656	
0	2	16	104	640	
0	2	24	216	1728	

The last step is to multiply the last vector with y:

1	4	20	112	656	0
0	2	16	104	640	0
0	2	24	216	1728	2368

This gives us the completed table and the total number of Euler Tours for $G(2,4)$, which is 2368.

We start backtracking from the rightmost cell (the one containing the number of Euler Tours):

- 2368 was the sum of 640 and 1728; we pick a number between 0 and 2368 and if it is smaller than 1728 we go to the cell containing 1728, otherwise we go to the cell containing 640. Suppose the random number we pick is 2000. Then we backtrack to the cell containing 640; this corresponds to the second transition class, $[l_0, r_1]$;
- 640 was the sum of the second column of A and the previous column, so $640 = 112 \cdot 2 + 104 \cdot 4 + 216 \cdot 0$; we pick a number between 0 and 640; if it is between 0 and $112 \cdot 2$ we pick the cell containing 112, if it is between $112 \cdot 2$ and $112 \cdot 2 + 104 \cdot 4$ we pick the one containing 104, otherwise we go to the other one (in this case there are actually only two choices because 216 has weight 0, so the probability to pick it is 0); suppose the random number is 114; this means we pick the first transition class, $[l_0, r_0]$
- we continue to do so, until we have reached the second column of the table.

Suppose the transition classes we chose for the 4 columns of the grid are: $[l_0, r_0]$, $[l_0, r_1]$, $[l_0, r_0]$, $[l_0, r_1]$.

We now need to choose the class of transition systems for each individual column.

For the first column suppose the previous class was $[l_0, r_0]$; looking in the transition table, we see that the corresponding class for $([l_0, r_0], [l_0, r_0])$ is $[l_0, r_0]$ - therefore the first column will have pairings of edges corresponding to this class. From the 4 possible pairings we pick one at random. Say the partial transition system for the first column is:

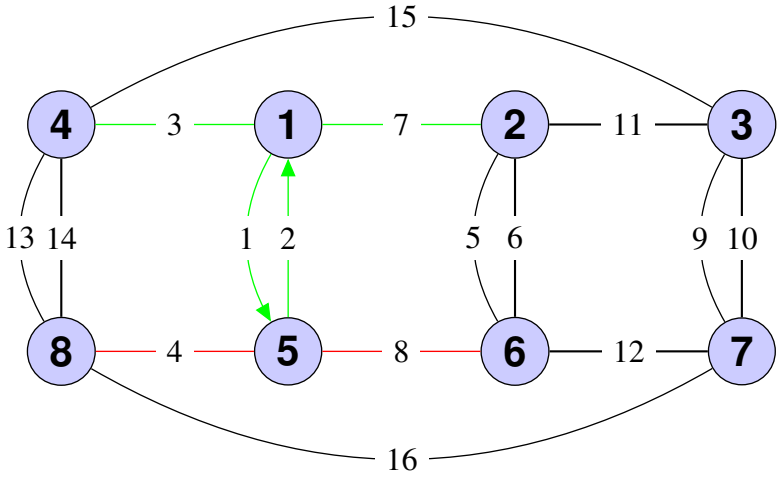


Figure 3.8: $G(2,4)$

In code this is represented by $\{\{3,1\},\{2,7\}\},\{\{4,8\},\{1,2\}\}$. The transition system of the graph is a hashmap where the keys are vertices and the value for each key is the pairing at that vertex. We add the array $\{\{3,1\},\{2,7\}\}$ with key 1 and $\{\{4,8\},\{1,2\}\}$ with key 5 to the hashmap.

Note: Even though there is a one-to-one correspondence between keys and values and the number of keys is known from the beginning (it is just the number of vertices) the hashmap was preferred over an array only because it made it easier to obtain the stored pairings for a certain vertex.

Looking again at the classes of transition systems we choose, which were $[l_0, r_0]$, $[l_0, r_1]$, $[l_0, r_0]$, $[l_0, r_1]$, we see that the class of the partial transition system for the first two columns should be $[l_0, r_1]$. Since the previous one was $[l_0, r_0]$, we look at the transition table in the cell corresponding to $([l_0, r_0], [l_0, r_1])$, where we have $[l_0, r_1]$. So we choose one of the 2 possible pairings for the $[l_0, r_1]$ class. Suppose we choose pairings [1][1]: $\{\{\{3,1\},\{2,7\}\},\{\{1,8\},\{4,2\}\}\}$, which, updated for column 2 is: $\{\{\{7,5\},\{6,11\}\},\{\{5,12\},\{8,6\}\}\}$. So far we have:

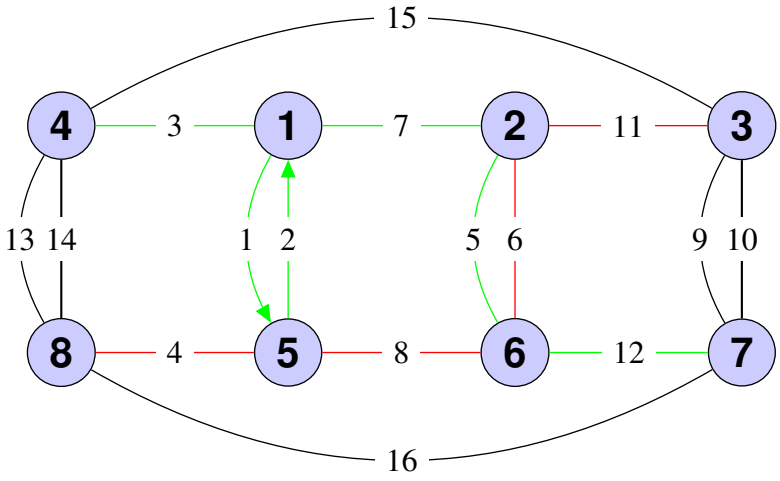


Figure 3.9: $G(2,4)$

We add to the transition system hashmap the values $\{\{7,5\},\{6,11\}\}$ with key 2 and the value $\{\{5,12\},\{8,6\}\}$ for key 6.

The class for the next column will be the one in cell $([l_0, r_1], [l_0, r_0])$, so $[l_0, r_1]$ and the class for the last column will be $[l_0, r_1]$ again. In the end, one possibility is that the tour will look like this:

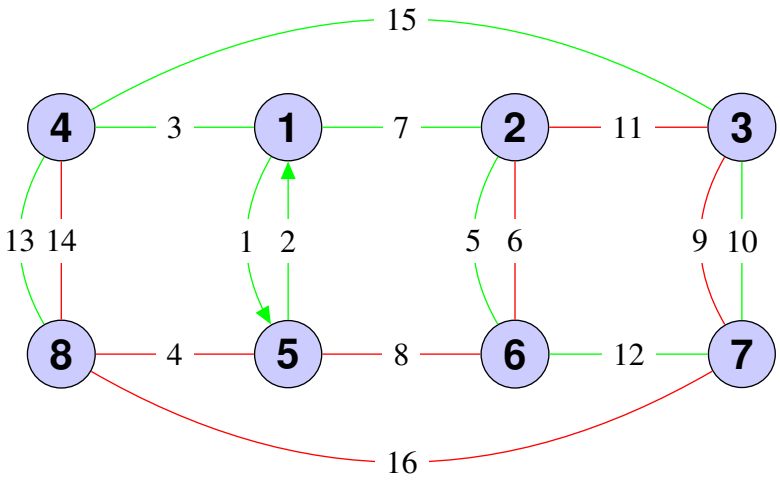


Figure 3.10: $G(2,4)$

where 14 and 3 should be paired, which combines the 2 paths into one. The final transition system is

Vertex		
1	3,1	2,7
2	7,6	5,11
3	11,9	10,15
4	15,13	14,3
5	4,8	1,2
6	6,12	8,5
7	9,16	12,10
8	13,4	16,14

Besides the transition system we also have a sequence of edges and vertices representation that we need in order to build the paths (we will explain in the next chapter why it is useful). For this we start by initializing an array of size (number of edges)+2(number of vertices) (which is $4 \cdot n \cdot m$ for $G(m,n)$) with the values 1,1 - we always start with the edge with index 1 and with the vertex with index 1 to account for tours that are equivalent under rotation and reversal. To make it easier to distinguish, we will represent vertices between brackets. Until now the edge vertex sequence is $1(1)$. Looking at the vertex (1) in the transition system we see that edge 1 is paired with 3, so we add this to the edge sequence: $1(1)3$. We then look at the neighbours of vertex (1) and check which ones have edges pairing with 3. In this example the neighbour is (4), with the pairing $\{14,3\}$. We update the edge sequence to: $1(1)3(4)14$. We keep doing that until the array is filled, at which point the sequence is:

$$1(1)3(4)14(8)16(7)9(3)11(2)5(6)8(5)4(8)13(4)15(3)10(7)12(6)6(2)7(1)2(5).$$

Chapter 4

Experimental analysis of the mixing time of the Kotzig Chain

We begin this chapter by reminding the reader of the technique we have used hoping get some insight into whether the Kotzig chain might be rapidly mixing. In the following sections we give details of the approach (Section 4.2), pointing to the implementation (Section 4.3). In the last section we go over an example in order to illustrate the algorithm.

4.1 Conductance and Canonical Paths

In Section 2.3.2 we discussed the "canonical paths" technique. To use it, we need to compute a quantity ρ which measures the edge loading for a set of paths between states in the state space:

$$\bar{\rho} = \bar{\rho}(\Gamma) = \max_e \frac{1}{Q(e)} \sum_{\gamma_{xy} \ni e} \pi(x)\pi(y)|\gamma_{xy}|,$$

where x and y are states in the state space, Γ is the set of paths, γ_{xy} denotes the path from x to y, $|\gamma_{xy}|$ is the length of the path and the maximum is over the edges of the state space (for a discussion of paths see Section 2.3.2). x and y must satisfy the detailed balanced condition

$$Q(x,y) = \pi(x) \cdot P(x \to y) = \pi(y) \cdot P(y \to x).$$

Note that, for Euler Tours, $\pi(x) = |\Omega|^{-1}$ (we are in the uniform distribution) and $P(x \to y) = \frac{1}{n}$ (because out of the 3 pairings possible at a vertex one is possible and one is prohibited, so we can only make one change at a time). Hence we can simplify the above expression for ρ to:

$$\bar{\rho} = \bar{\rho}(\Gamma) = \max_e \frac{n}{|\Omega|} \sum_{\gamma_{xy} \ni e} |\gamma_{xy}|.$$

As Jerrum and Sinclair observed [11], for the Markov chain to be rapidly mixing it needs to have no "bottlenecks" - we need to find a set of paths Γ such that $\bar{\rho}(\Gamma)$ is not too big. We experimentally evaluated conductance over a series of randomly generated paths of Euler Tours, inspired by Tetali and Vempala's approach[18]. They were trying to prove that the Kotzig Chain was rapidly mixing, but some of their assumption were false and the proof could not be fixed. In the next section we will describe in detail how they built the paths, we will point out the flaws in their proof and present our adapted experimental set-up.

4.2 Building the paths - Theory

Tetali and Vempala [18] attempted to prove that the Kotzig Chain is rapidly mixing by using the canonical paths argument due to Jerrum and Sinclair [11] (for a description of Kotzig moves and the Kotzig Chain please refer back to Section 2.4.2).

Kotzig and Abrham [1] observed that given two randomly sampled tours A and B we are certain we can build a path from A to B using Kotzig moves and formulated the following theorem:

Theorem 2. *[1] Let A and B be two Euler tours of an Eulerian multigraph G. Then there exists a finite sequence of* κ*-transformations at the vertices of G which transforms A into B.*

This proves that the set space is connected, which is needed when trying to sample from a Markov Chain. However, to prove that the chain is rapidly mixing we need to find a good set of paths and a rigorous proof of high conductance. Tetali and Vempala attempted this with no success, but let us show how they built the paths.

The Kotzig moves are local transformations that change the pairing of the edges at a vertex. Therefore, a useful way to represent Euler tours of the grids is by their transition systems - for each vertex v of the tour we write the set $\{\{e,e'\},\{f,f'\}\}$, where e,e',f,f' are the 4 edges incident at vertex v, grouped according to their pairing (for the definition of a transition system please refer to Section 2.4.1). By this representation, on one visit the tour enters v at e and leaves at e' (or vice versa) and on its other visit it enters along f and leaves along f' (or vice versa).

We can easily build the transition system of any tour, hence we can then compare the pairings of edges at each vertex of any pair of tours A and B and take a note of the disagreement vertices. By the above theorem, using a number of κ-transformations, we can fix the transition system at each of the vertices in the disagreement set (where

Note:The content of this section is based on [4].

by "to fix" we mean to change the pairing of edges at a vertex in A into the same pairing for that vertex in the final tour B). So every time we make a move at a vertex, the current Euler Tour is transformed into another, whose transition system differs from the transition system of the previous tour at exactly one vertex (the one where the move was performed).

In general, we denote by $TS(T,v)$ the pairing given by the transition system of a tour T at vertex v. Suppose that the 4 incident edges of the vertex v of an Euler Tour of a 4-regular grid are: e, e', f, f'. Then the only 3 possible pairings of the edges at vertex v are:

1. $TS(T,v) = \{\{e,e'\},\{f,f'\}\}$
2. $TS(T,v) = \{\{e,f\},\{e',f'\}\}$
3. $TS(T,v) = \{\{e,f'\},\{e',f\}\}$.

If one of these pairings is the current pairing at a vertex, out of the two remaining ones only one can be achieved by a κ-transformation, depending on which order the edges are visited in the tour. In a edge sequence representation of the tour the pairing $TS(T,v) = \{\{e,e'\},\{f,f'\}\}$ will correspond to one of:

- $...e(v)e'...f(v)f'...$, or the inverse $...f'(v)f...e'(v)e...$
- $...e(v)e'...f'(v)f...$, $...f(v)f'...e'(v)e...$
- $...e'(v)e...f(v)f'...$, $...f'(v)f...e(v)e'...$
- $...e'(v)e...f'(v)f...$, $...f(v)f'...e(v)e'....$

Let us consider now how and when the moves are performed to build the path from tour A to tour B. We already know that we can build a set of disagreement vertices that we wish to fix. Suppose we are trying to fix the vertex v in tour A such that it has the same pairing as in tour B. Say that $TS(A,v) = \{\{e,e'\},\{f,f'\}\}$ and that the edge sequence is of the form: $...e(v)e'...f(v)f'....$ To fix this vertex we distinguish between the following cases, which depend on the pairing of v at B and the sequence of edges:

1. $TS(B,v) = \{\{e,f\},\{e',f'\}\}$ and the edge sequence at v is either $...e(v)f...e'(v)f'...$ or $...f'(v)e'...f(v)e...$ - to obtain the same pairing in tour A we only need to reverse the sequence of edges traversed in between the two visits at vertex v, so the ones between e' and f in A; the transition system at all vertices except v will not change.
2. $TS(B,v) = \{\{e,f\},\{e',f'\}\}$ but the edge sequence is none of the ones above or $TS(B,v) = \{\{e,f'\},\{e',f\}\}$ - we will need a helper vertex u that will make the required κ-transformation available; the transition system will be transformed to the one of B at no more than 2 vertices (the initial one and the helper) and otherwise will remain the same.

In the first case we say that the desired move at v is *available*, while in the second case the move is *prohibited*.

The existence of a helper vertex is proved by the following lemma:

Lemma. *[18] Let A and B be two Euler tours of a 4-regular graph G. Suppose* $TS(A,v) \neq TS(B,v)$*, and the move at vertex v is not available, then*

1. there exists $u \neq v$ *such that* $TS(A,u) \neq TS(B,u)$*, and*

2. in at most 3 κ*-transformations (on A) we can transform A to* A' *so that* $TS(A',u) = TS(B,u)$*,* $TS(A',v) = TS(B,v)$*, and* $TS(A',w) = TS(A,w)$*, for* $w \neq u,v$*.*

Proof. Let us argue the first part by contradiction. Suppose the disagreement set is empty, so the transition system of A is the same as the one of B except at vertex v. But if A differs from B only at v and the necessary move at v is not allowed, then one of A and B cannot be an Euler Tour.

For the second part, suppose there exists at least one more vertex that needs to be fixed. Tetali and Vempala [18] observed that a κ-transformation at a vertex will either keep the rest of the transition system unchanged or, for a vertex v, it will make the prohibited pairing allowed - this happens when u and v are interleaved. Suppose that in our disagreement vertices set there are no vertices that interleave v. This means that we can fix all of them without changing the transition system at v, which will leave v the only vertex in the disagreement set - we have already seen this is not possible. Therefore, there must be at least one vertex in the disagreement set that interleaves v. If the move at u is available then, because it interleaves v, it will make the prohibited move at v (the one we needed) available. If the move at u is prohibited, then making a κ-transformation at v will make the desired move at u available, which in turn will make the fixing move at v available. □

This lemma and its proof tell us a couple of very important things about the helper vertex u. First, u needs to be part of the disagreement set and by the proof of the lemma, if a helper is needed there will always be one in this set. Secondly, from the disagreement vertices we need to choose u such that it interleaves v, the vertex we were initially trying to fix - without this condition the helper would not be able to actually help ”unlock” the desired move on v. Once we have identified the helper, there are two possible situations:

1. the move at u is available - in this case we first make a κ-transformation at u and then the fixing κ-transformation at v
2. the move at u is prohibited - we first make a κ-transformation at v, which will allow us to make the κ-transformation at u, which in turn will make the desired κ-transformation at v available; in the end the new tour will have both u and v fixed.

Another observation we can make based on the previous lemma is that there may be many possible paths from a tour A to a tour B. It is true that at any vertex, out of the two possible moves, only one can actually be carried out if we still want to obtain an Euler Tour. However, whenever we need a helper vertex we can choose any vertex that interleaves the vertex to be fixed and is still in the remaining disagreement set, and depending on which vertex we choose, the next tour may take various forms.

Tetali and Vempala's approach was to consider a pairing of the disagreement vertices, in such a way as to give an order in which vertices would be fixed. Canonical paths having this fixed ordering have been really useful to prove polynomial bounds on ρ, so this was a desirable property for Tetali and Vempala's paths. However, they failed to consider a couple of things. One would be the above observation - a vertex can have many possible helpers and a helper may fix multiple vertices. Moreover, we have seen that making a κ-transformation at some vertex will interchange the allowed and prohibited transformations for all interleaving vertices; so if any of these needed to be fixed with a helper, after one transformation they may not need a helper anymore. Therefore, the relationship between vertices and helpers and the order in which they propose to carry out the transformations established at the beginning may be changed after a number of moves.

For these reasons we considered the following changes: we order the vertices by index and sort the disagreement set according to that order; whenever we need a helper, choose the one that has the lowest index. Update the transition system and delete the fixed vertices from the disagreement set, updating the counts of the space state edges accordingly. The approach is presented in full in the next section.

4.3 Building the paths - The Algorithm

We now discuss what steps we took in building the paths and how we have implemented them. Based on this, an illustrative example will be given in the next section.

Given two tours A and B, the first step is to build the disagreement set and we do this by comparing the transition system at each vertex in the tours, in order. This will give us a sorted set of the difference vertices, which we will fix one by one.

As we have seen, in order to check whether a κ-transformation at some vertex is available, we need to know the order in which edges are visited (Note: an Euler tour does not have a predefined order, but we will consider the relative order of the 4 edges of a vertex amongst themselves). For this, besides the transition system representation we also store the tours as sequences of vertices and edges. Therefore the two representations we used for tours are:

1. a sequence of edges and vertices of the form : $e_1, v_1, e_2, v_2, ...$, where e_i and v_i are the indices of the edges and vertices of the grid; for this purpose we are using an array in which indices of vertices are stored at odd positions and indices of edges at even positions. We will always start with vertex 1 and edge 1 to account for the fact that some tours are equivalent under rotation and reversal;
2. a transition system - for each vertex we write the set $\{\{e, e'\}, \{f, f'\}\}$ that gives us the pairings of the edges incident at that vertex. We store this as hashmap with keys the indices of vertices and values 2×2 arrays where each row corresponds to a pairing. Even though the hashmap has constant size, we have chosen this data structure to be able to easily access the pairings, compare and change them.

Note: The reason why we did not choose a more compact representation - for example an ordered version of the transition system, where each pairing $\{\{e,e'\},\{f,f'\}\}$ is written according to the order in which the edges e,e',f,f' appear in a tour - is that making a κ-transformation preserves only the pairings of the other vertices and not the order in which they are visited. Changing the order in which vertices are visited also changes the order of the edges, and this influences the available moves for other vertices. Having vertices and edges written explicitly makes it easier to keep track of κ-transformations and availability.

Once we have the two representations for each of the two sampled tours we start building the path. Given the transition systems represented as hashmaps, we iterate through every key (each representing a vertex of the grid) and check if the associated pairings of vertices are the same in the two tours, disregarding the order. So we consider the pairings $\{\{e_1,e_3\},\{e_2,e_7\}\}$ and $\{\{e_2,e_7\},\{e_3,e_1\}\}$ equivalent, but $\{\{e_1,e_3\},\{e_2,e_7\}\}$ and $\{\{e_1,e_2\},\{e_3,e_7\}\}$ different. After iterating through all vertices we have filled the set of disagreement vertices (represented as a sorted ArrayList) with the indices of all the vertices where the pairings are different in the two tours. As we build the path between the two tours we will keep deleting from this set those vertices that have been fixed after one or more κ-transformations.

At every step in the path we consider the lowest indexed vertex in the disagreement set, call it v.

- We check whether the κ-transformation at v needed to fix the transition system is available - which means we are in the case: $TS(A,v) = \{\{e,e'\},\{f,f'\}\}$ and A has edge sequence $...e(v)e'...f(v)f'...$, $TS(B,v) = \{\{e,f\},\{e',f'\}\}$ and B has edge sequence $...e(v)f...e'(v)f'...$. If so, we reverse everything between e' and f in A and change the pairing of v in A to the one in B. We delete the fixed vertex from the disagreement set and move on to the next one.
- If the κ-transformation is prohibited, we start looking for a helper vertex u. For this we need to choose the vertex with the smallest index from the disagreement set that interleaves v in the tour - call this vertex u.
  - If the κ-transformation at u is available, we make this transformations (updating the sequence of edges and the transition system) and then make the κ-transformation at v (again, updatind the sequences of edges and transition system).
  - If the κ-transformation at u is not available, we make the following 3 moves: first make a κ-transformation at v (which is not the one we need to fix the transition system, but a different one), then check if we can make the desired κ-transformation at u, after which we make another κ-transformation at v, this time the one we needed.

With every vertex that is fixed we update the set of disagreement vertices (either by deleting v or v and u) and also we increase the count for the space states involved in the κ-transformations. Whenever we make a move at a vertex we transition from one Euler Tour to another. Normally, we would have to increase the count of each state with a value corresponding to the length of the path. But we can only know this value once

we have finished building the path. However, since for each vertex in the disagreement set we either need one, two or three κ-transformations and each of them corresponds to a new state of the path, we can assume that we will need between d and $1.5 \cdot d$ moves (where d is the size of the disagreement set). We have chosen to update the count of each state involved by $1.25 \cdot d$. One thing to note is that, instead of having states as (A_i, A_{i+1}) where A_i and A_{i+1} are tours, we store them as (A_i, v) where v is the vertex where we have made the move to get from tour A_i to A_{i+1}, because, given the length of a tour even for a small grid, we need to store as little as possible. For this reason, whenever we update the hashmap of counts we do so for both (A_i, v) and (A_{i+1}, v). Suppose we are currently at tour A_i an are trying to fix vertex v. If the fixing move is direct, we would update states (A_i, v) and (A_{i+1}, v) with the value $1.25 \cdot d$ (either add this value to the existent count or add the state to the hashmap with this initial value). If the move is not direct, depending on how many moves we need, we either first update states (A_i, u) and (A_{i+1}, u), where u is the helper and then states (A_{i+1}, v) and (A_{i+2}, v) or, in this order: (A_i, v), (A_{i+1}, v), (A_{i+1}, u), (A_{i+2}, u), (A_{i+2}, v) and (A_{i+3}, v). This hashmap will be used throughout the entire simulation, maintaining the count for all the pairs of tours generated.

We will next present all of the above steps with the help of 2 sampled tours.

4.4 Illustrative example

In this section we will work through an example to illustrate every step involved in the algorithm implemented to build a path. We have chosen an example such that all types of moves with the different possible cases will be present. As before, vertices are written between brackets. We will be working with tours on the $(2,4)$ grid.

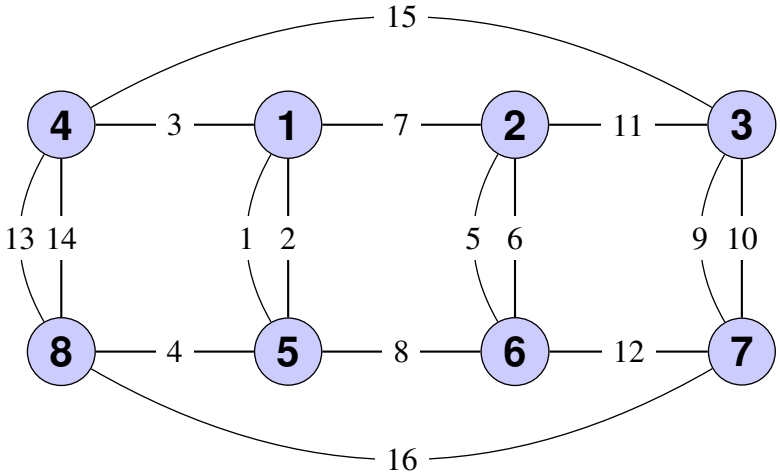


Figure 4.1: (2,4)Grid

Suppose tour A is:

1 (1) 3 (4) 13 (8) 16 (7) 9 (3) 11 (2) 5 (6) 6 (2) 7 (1) 2 (5) 4 (8) 14 (4) 15 (3) 10 (7) 12 (6) 8 (5)

and tour B is:

1 (1) 3 (4) 14 (8) 13 (4) 15 (3) 11 (2) 7 (1) 2 (5) 4 (8) 16 (7) 10 (3) 9 (7) 12 (6) 6 (2) 5 (6) 8 (5)

with the corresponding transition systems:

Vertex	Tour A	Tour B
(1)	$\{\{1,3\},\{2,7\}\}$	$\{\{1,3\},\{2,7\}\}$
(2)	$\{\{5,11\},\{6,7\}\}$	$\{\{5,6\},\{7,11\}\}$
(3)	$\{\{9,11\},\{10,15\}\}$	$\{\{9,10\},\{11,15\}\}$
(4)	$\{\{3,13\},\{14,15\}\}$	$\{\{3,14\},\{14,15\}\}$
(5)	$\{\{1,8\},\{2,4\}\}$	$\{\{1,8\},\{2,4\}\}$
(6)	$\{\{5,6\},\{8,12\}\}$	$\{\{5,8\},\{6,12\}\}$
(7)	$\{\{9,16\},\{10,12\}\}$	$\{\{9,12\},\{10,16\}\}$
(8)	$\{\{4,14\},\{13,16\}\}$	$\{\{4,16\},\{13,14\}\}$

Then the disagreement set is $\{(2),(3),(4),(6),(7),(8)\}$ and say the HashMap of states is currently empty; the value we will add at each update is $1.25 \cdot 5 = 6.25$, because the size of the disagreement set is 5.

We start by trying to fix vertex (2) - we need to check the availability of the needed move. The corresponding edges for vertex (2) are:

$$A : 11(2)5...6(2)7$$
$$B : 11(2)7...6(2)5$$

therefore the direct move is not available (we should have had $11(2)6...5(2)7$ or the inverse in B).

Since the move is not direct, we will start looking for a helper. The only vertices that interleave (2) in A are (5) and (6). Only (6) is in the disagreement set, so this is the only possible helper.

Fixing helper (6) - we need to check the availability of the direct move. The corresponding edges for vertex (6) are:

$$A : 5(6)6...12(6)8$$
$$B : 12(6)6...5(6)8$$

so the direct move is not available. This means we are in the 3 moves case described in the previous section - we need to make the available move at (2), which will allow us to do the desired move at (6), which in turn will allow the fixing move at (2). So first let us we make the available move at (2) in A, which will give A_1 with transition system:

$TS(A_1,(2)) = \{\{5,7\},\{6,11\}\}$ and $TS(A_1,(v)) = TS(A,(v))$ for $v \neq 2$

and edge sequence:

1 (1) 3 (4) 13 (8) 16 (7) 9 (3) 11 (2) 6 (6) 5 (2) 7 (1) 2 (5) 4 (8) 14 (4) 15 (3) 10 (7) 12 (6) 8 (5).

We now add states $(A,(2))$ and $(A1,(2))$ to the HashMap of counts, each with value 6.25.

Observe that we can now do the needed move at (6):

$$A_1 : 6(6)5...12(6)8$$
$$B : 12(6)6...5(6)8$$

This move will give A_2 with transition system

$TS(A_2,(6)) = \{\{6,12\},\{5,8\}\}$ and $TS(A_2,(v)) = TS(A_1,(v))$ for $v \neq 6$

and edge sequence:

1 (1) 3 (4) 13 (8) 16 (7) 9 (3) 11 (2) 6 (6) 12 (7) 10 (3) 15 (4) 14 (8) 4 (5) 2 (1) 7 (2) 5 (6) 8 (5).

We need to update the counts of states $(A_1,(6))$ and $(A_2,(6))$, by adding them to the HashMap with value 6.25.

The previous move fixes helper (6) and allows us to fix (2) - the move at (2) is now available. The transition system of A_3 is:

$TS(A_3,(2)) = \{\{5,6\},\{7,11\}\}$ and $TS(A_3,(v)) = TS(A_2,(v))$ for $v \neq 2$

and edge sequence:

1 (1) 3 (4) 13 (8) 16 (7) 9 (3) 11 (2) 7 (1) 2 (5) 4 (8) 14 (4) 15 (3) 10 (7) 12 (6) 6 (2) 5 (6) 8 (5)

We will update the counts of states $(A_2,(2))$ and $(A_3,(2))$ by adding them to the HashMap with value 6.25.

Now that we fixed vertices (2) and (6) we remove them from the disagreement set, which is now: $\{(3),(4),(7),(8)\}$.

The next one to be fixed is vertex (3). The needed move is not available so we again look for a helper. Looking at the interleaving vertices and the disagreement set we see that the lowest index vertex that appears in both is (4). We can do the direct move at (4) which allows us to do the fixing move at (3). Each time we make a move we add the relevant entries in the HashMap. We delete (3) and (4) from the disagreement set and move on to the next vertex. (7) and (8) can both be fixed by direct moves.

After this we move on to the next pair, going through the same steps, just that this time not all of the states will be added to the HashMap, because we may have seen them already - therefore if a state is already there we will be update it by adding the relevant value to the existent count (1.25·the size of the new disagreement set).

4.5 Results

We have run the path building algorithm on grids of sizes $(2,n)$ and $(3,n)$ with n between 5 and 10. We have sampled 10^6 pairs for the $(2,n)$ grids and 10^5 pairs for the $(3,n)$ grids. We will account for this in our calculation of the estimate of ρ by multiplying with $(|\Omega| * (|\Omega|-1))/P$. The expression we had before for ρ was:

$$\bar{\rho} = \bar{\rho}(\Gamma) = \max_e \frac{n}{|\Omega|} \sum_{\gamma_{xy} \ni e} |\gamma_{xy}|.$$

The estimate will therefore be:

$$\rho_{estimate}(\Gamma) = \frac{n(|\Omega|-1)}{P} \max_{e} \sum_{\gamma_{xy} \ni e} |\gamma_{xy}|.$$

Note that we are also estimating $\sum_{\gamma_{xy}}$ when we approximate the length of the path with $1.25d$, where d is the size of the set of the disagreement vertices between two tours. Adding the actual length of the path would have meant that all updates of the states had to be done after the path was constructed, which would have made computation even more difficult.

These are the results for $\rho_{estimate}$:

$(2,n)$	5	6	7	8	9	10
tours	$1.6 \cdot 10^4$	$1.1 \cdot 10^5$	$7.9 \cdot 10^5$	$5.3 \cdot 10^6$	$3.5 \cdot 10^7$	$2.3 \cdot 10^8$
$\rho_{estimate}$	$2.5 \cdot 10^2$	$5.5 \cdot 10^2$	$1.5 \cdot 10^3$	$7.1 \cdot 10^3$	$3.7 \cdot 10^4$	$2.2 \cdot 10^5$

which we approximated to:

$(3,n)$	5	6	7	8	9	10
tours	$1.7 \cdot 10^6$	$2.8 \cdot 10^7$	$4.5 \cdot 10^8$	$7.2 \cdot 10^9$	$1.1 \cdot 10^{11}$	$1.7 \cdot 10^{12}$
$\rho_{estimate}$	$6 \cdot 10^3$	$1.4 \cdot 10^5$	$2.4 \cdot 10^6$	$4.6 \cdot 10^7$	$6.8 \cdot 10^8$	$1.3 \cdot 10^{10}$

We believe that more pairs of tours should be sampled in order to get a better idea whether the chain might have high conductance.

Chapter 5

Conclusions

In this chapter we will review the main achievements and describe what areas can be improved.

Our project aimed to experimentally study whether the Kotzig Chain may be rapidly mixing, which would indicate that an efficient approximation algorithm for the number of Euler Tours on 4-regular graphs could be found. For this there were three main objectives: sampling Euler Tours on graphs, building the paths between pairs of tours, analyse the conductance based on the experimental results. A lot of effort has been put into understanding the required readings and filling in the gaps of existent results. The sampling and building the paths algorithms have been implemented and also the tools to get insight on the values for the edge loading. More time could we put into analysing the results. However, other (unexpected) contributions have been made, which were fixing the transfer matrix for the $(3,n)$ grid and showing that we can still use the "transfer matrix" approach to sample Euler Tours.

For further work, besides running the algorithm with more samples, some effort could be put into finding a way to represent the space states that would make the computation more efficient. Right now, we are using the sequence of edges representation annotated with the vertex and storing this in a hashmap. One thing we could do is to create an array of hashmaps that would group the states according to the vertex where the change was made. Finding ways to sample more pairs in a shorter time would be the main improvement to the coding part of the project.

Bibliography

[1] J. Abrham and A. Kotzig, Transformations of Euler Tours, *Annals of Discrete Mathematics 8*, pp. 65-69, 1980

[2] G. Brightwell and P. Winkler, Counting Eulerian circuits is ♯P-complete, *Proceedings of the Second Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2005)*, pp. 259-262, 2005

[3] P. Creed, Counting and Sampling Problems on Eulerian Graphs, *PhD thesis, University of Edinburgh*, 2010

[4] M. Cryan, *Personal communication*

[5] M. Cryan, P.Chebolu, R. Martin, Exact counting of Euler Tours for Graphs of Bounded Treewidth, *CoRR*, 2013

[6] M. Dyer, Approximate Counting by Dynamic Programming, *Proceedings of the 35th ACM Symposium on Theory of Computing*, pp.693-699, 2003

[7] Mordecai J. Golin, Yiu-Cho Leung, Yajun Wang, and Xuerong Yong, Counting Structures in Grid Graphs, Cylinders and Tori Using Transfer Matrices: Survey and New Results, In *Proceedings of the 2nd Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2005)*, pp. 250-258

[8] X. R. Yong, M. J. Golin, Algebraic and combinatorial properties of the transfer matrix of the 2-dimensional (1,∞)-runlength limited constraint, *Findings of The 2002 IEEE International Symposium on Information Theory*, pp. 354, 2002.

[9] M. Jerrum, The computational complexity of counting, Proceedings of the International Congress of Mathematicians, *Zürich 1994, Birkhuser, Basel*, 1407-1416, 1995

[10] M. Jerrum, Review MR1822924 (2002k:68197) of [13], *MathSciNet*, 2002

[11] M. Jerrum and A. Sinclair, The Markov Chain Monte Carlo Method: An Approach to Approximate Counting and Integration, *PWS Publishing*, pp.482-520, 1996

[12] M. Jerrum, L. Valiant, and V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoretical Computer Science, 43(2-3)* pp.169-188, 1986

[13] G. Kirchhoff, Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird, *Annalen der Physik, vol. 148,* pp.497-508, 1847

[14] A. Kotzig, Eulerian lines in finite 4-valent graphs and their transformations, In: *P.Erdös and G. Katona, Eds., "Theory of Graphs," Proc. of the Colloq. held at Tihany, Hungary*, 1966

[15] M. Mihail and P. Winkler, On the Number of Eulerian Orientations of a Graph, *Algorithmica 16*, pp. 402-414, 1996

[16] H. Rogers, Theory of Recursive Functions and Effective Computability, *McGraw-Hill Book Company*, 1967

[17] A. Sinclair, Improved bounds for mixing rates of Markov chains and multicommodity flow, *Combinatorics, Probability and Computing*, pp.351-370, 1992

[18] P. Tetali and S. Vempala, Random Sampling of Euler Tours, *Algorithmica 30*, pp. 376-385, 2001

[19] L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science 8*, pp. 189-201, 1979

[20] Eric Vigoda, Lecture notes from Foundations of Markov chain Monte Carlo methods, *University of Chicago, USA, Lecture 1* March 29,2002