Generation of random graphs with applications to complex networks

Srivatsan Vasudevan
University of Windsor

Generation of random graphs with applications to complex networks

By

Srivatsan Vasudevan

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2024

Generation of random graphs with applications to complex networks

by

Srivatsan Vasudevan

APPROVED BY:

M. S. Monfared
Department of Mathematics and Statistics

Jianguo Lu
School of Computer Science

A. Mukhopadhyay, Advisor
School of Computer Science

12 June, 2024

ABSTRACT

Given a sequence d of n positive integers, $n-1 \geq d_1 \geq d_2 \geq \ldots \geq d_n \geq 0$, different sets of necessary and sufficient conditions have been proposed for the graphicality of such a sequence. When such a graph exists, we can use 2-switches of pairs of independent edges to transform one such graph G into another graph G' with the same degree sequence. This was proved by various authors. In this thesis we address the question whether a given graph G is at all switchable. Indeed, we show that unswitchable graphs are a proper subclass of split graphs, and exploit this fact to propose efficient algorithms for the recognition and generation of unswitchable graphs. This question of unswitchability is important as switching has been tried as a mechanism for generating a random graph with a given degree sequence or for generating new ones, preserving properties like simplicity or connectedness (this is known as constrained switching).

In the second part of this thesis, motivated by the application to the area of so−called complex networks (examples are: protein−protein interaction networks, social networks, metabolic networks etc.), the statistical properties of province-wise transportation networks of Canada are studied and compared with two random graphs, generated using the Erdos-Renyi model and the Configuration model. A method to extract transportation network and network resilience two key practical contributions are discussed in depth.

Finally, taking cue from the configuration model, in an appendix we discuss an implementation that generates a directed graph uniformly at random, using a Markov Chain Monte Carlo (MC−MC) method.

DEDICATION

To all the people who have supported me in my whole academic journey.

ACKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervisor Dr. A. Mukhopadhyay, his continuous interaction helped me gain valuable research experience. His ideas and approach to a problem created an intrigue in this field and was the driving force to help me present this research.

I want to thank my committee members M. S. Monfared and Dr Lu for their valuable time and input on this thesis.

I want to thank the department of computer science and its computer science faculty for imparting the knowledge necessary for this work. Finally, to my friends and family who have always believed in me throughout this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Graph theory is a fundamental area of mathematics with extensive applications in various fields such as computer science, biology, social sciences, and transportation. It provides powerful tools to model and analyze complex networks, which are crucial for understanding the underlying structure and dynamics of real-world systems. In this thesis, two problems in the domain of complex networks one a theoretical problem and the other a practical problem are discussed.

1.1 Preliminaries

A graph is a set of vertices and edges. Graphs can be categorized into directed and undirected graphs based on whether the edges have a direction or not. Figures 1.1.1 show an example of a directed and an undirected graph.

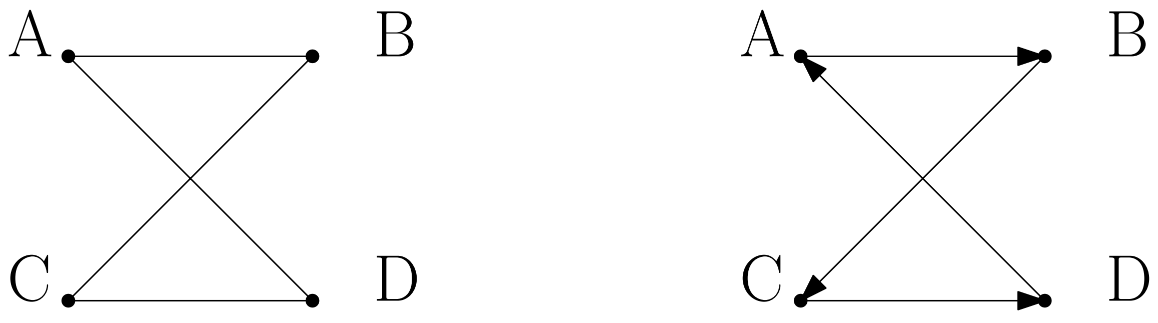


Fig. 1.1.1: Sample Directed and Undirected Graph

A graph can represented in the form of an adjacency matrix. For a graph with n vertices, an adjacency matrix is an $n * n$ matrix with each cell denoting the relation on how the vertices are connected. Figures 1.1.3 and 1.1.2 are the adjacency matrices of the graphs in Figure 1.1.1.

	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0

Fig. 1.1.2: Representation of undirected graph in fig 1.1.1 as adjacency matrix

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

Fig. 1.1.3: Representation of directed graph in fig 1.1.1 as adjacency matrix

1.2 Graph Terminologies

Degree of node: For an undirected graph, the degree of a node represents the number of connections that node has.

In-degree (out-degree) of a node: In the case of a directed graph, the in-degree (out-degree) of a node is a count of the number of edges that come into (leave) the node. We use in-edges (out-edges) edges as shorthand for incoming (outgoing) edges.

Multigraph: It is a graph in which self-loops and multiple edges are allowed. Figure 1.2.2 is an example of a multigraph.

Simple graph: A simple graph has no multiple edges and self-loops. Figure 1.2.1 is an example of a simple graph

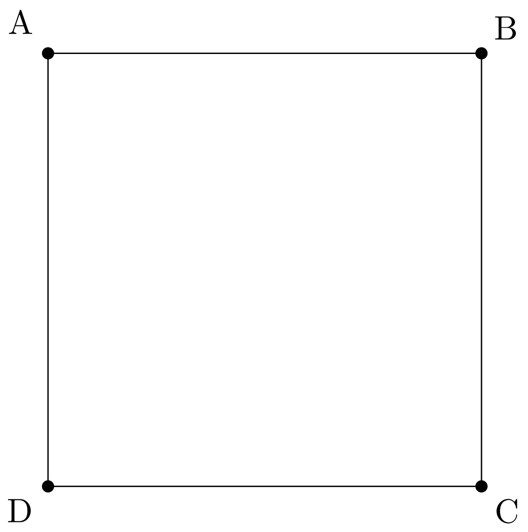


Fig. 1.2.1: Example of a simple graph

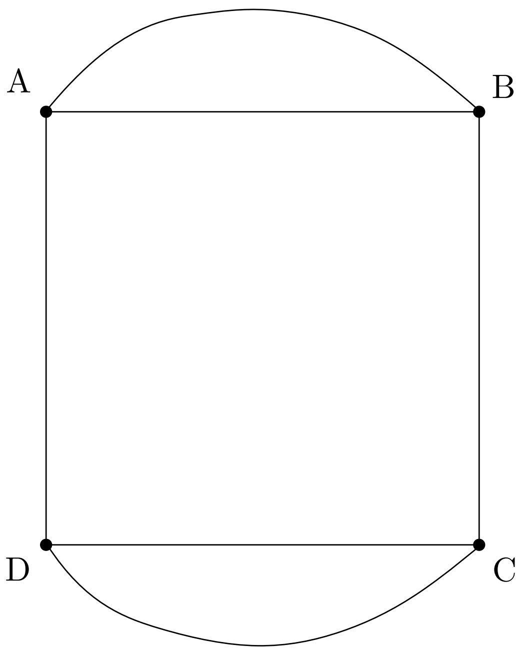


Fig. 1.2.2: Example of a multi graph

Degree Sequence: A list of non-negative integers that represent the degree of each node in the graph in that order is called a degree sequence. For example, the degree sequence of the undirected graph in figure 1.1.1 is [2, 2, 2, 2]. For a directed graph the degree sequence is split into in-degree sequence and out-degree sequence. For the directed graph in figure 1.1.1 the in-degree and out-degree sequences are [1, 1, 1, 1] and [1, 1, 1, 1] respectively.

Edge-Switching: This is the process of switching any two independent edges (two edges that do not have an end-point in common) $\{u, v\}$ and $\{x, w\}$ of a graph G to the edges $\{u, x\}$ and $\{v, w\}$ or $\{u, w\}$ and $\{v, x\}$ (Figure 1.2.3)

1.3 Prior Work

Edge-switching is a mechanism that helps to move from one realization graph to another without affecting the degree sequence of the graph. This is a tried mechanism for generating a random graph with a given degree sequence. Taylor [17] showed that we can move from a connected graph G to another graph G' through switching,

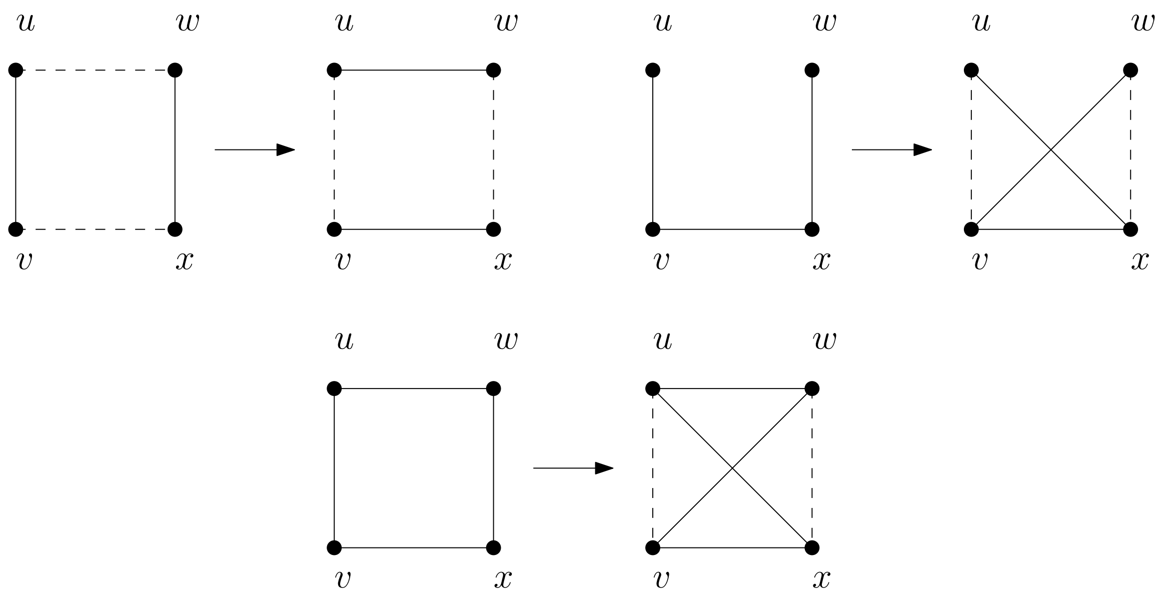


Fig. 1.2.3: Example of Edge Switching

preserving the connectivity of G. Lowcay et al. [12] provided a constructive proof for constrained switching. However, not all graphs are switchable and in this thesis, we have addressed this problem.

1.4 Thesis Organisation

This thesis is divided into four chapters.

Chapter 1: In the first chapter, we define graphs, and some associated terminologies used throughout. We discuss some prior works on degree sequence and edge switching and introduce the class of unswitchable graphs

Chapter 2: In the second chapter, we explore the class of unswitchable graphs in-depth and provide algorithms for identification for recognizing an unswitchable graph given a realization of the graph and generation of unswithcable graphs given the number of vertices required in the graph.

Chapter 3: In the third chapter, a practical application of a complex network is discussed. The transportation network is chosen as the subdomain. First A method to extract road networks of 10 provinces of Canada is discussed. Then two random graphs with the same number of nodes and edges as the province road network are generated using the Erdos Renyi model and the Configuration model. A comparative analysis both among provinces and between province networks and random graphs is done and the results are discussed.

Chapter 4: In the fourth chapter, contains the conclusion and future work for this thesis. Taking a cue from the configuration model to generate a random graph, a method to generate a directed graph with given marginals using the Markov Chain Monte Carlo (MC-MC) method is furnished in the appendix.

CHAPTER 2

Recognizing and generating unswitchable graphs

2.1 Introduction

Let $G = (V, E)$ be a simple graph on the vertex set $V = \{v_1, v_2, \ldots, v_n\}$. Let d_i be the degree of v_i. Assume without loss of generality that $n - 1 \geq d_1 \geq d_2 \geq \ldots \geq d_n \geq 0$. There may exist many other graphs G with the same degree sequence, making it a many-to-one mapping.

Let the edges $\{u, v\}$, $\{w, x\}$ of G be independent (this means that the edges do not have an end-point in common). For each of the three ways the edges can be independent, we can obtain another graph G' with the same degree sequence by means of a 2-switch as shown in Figure 2.1.1, in pairs from left to right and top to bottom, where the dashed lines show the replacement edges.

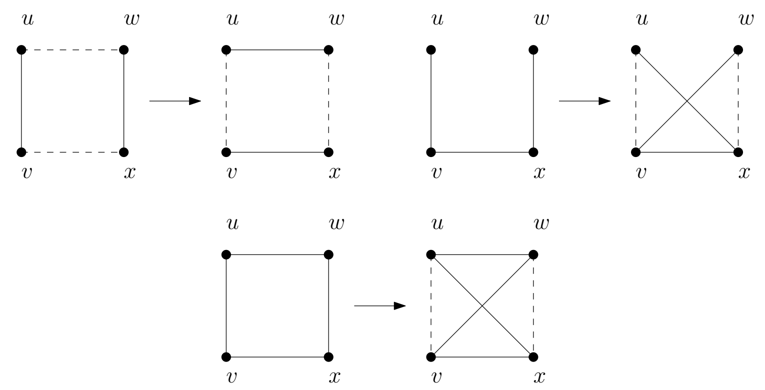


Fig. 2.1.1: Graph G' obtained from G by a 2-switch

A graph G is said to be *unswitchable* if it cannot be reduced to another graph H with the same degree sequence by edge-switching. In this paper, we propose an algorithm for recognizing unswitchable graphs that exploits the relationship of this class of graphs to the class of split-graphs.

To motivate the significance of the concept of edge-switching we dicuss an application in the next section.

2.2 An application of edge-switching

Given a graph G, it is easy to obtain its degree sequence d. However, given d with the d_i's in non-increasing order, the question whether there exists a graph whose degree sequence is d has spawned a lot of research. We begin with the following definition.

Definition 1 *The sequence d is graphical if there exists a graph G such $d_G(v_i) = d_i$ for $i = 1, \ldots, n$, where $d_G(v_i)$ denotes the degree of the vertex v_i in G.*

Both Hakimi [7] and Havel [9] are credited with the following result.

Theorem 1 *Let $n \geq 2$ and $d_1 \geq 1$. The sequence d is graphical if and only if the sequence $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \ldots, d_n$, arranged in nonincreasing order, is graphical.*

To prove this result we need a definition and prove two other results.

If a graph H can be obtained from a graph G by a finite sequence of 2-switches we indicate this reduction by the notation $G \overset{2s}{\Longrightarrow} H$. Berge [1] proved that:

Theorem 2 *Two graphs G and H on a common vertex set V satisfy $d_G(v) = d_H(v)$ for all $v \in V$ if and only if $G \overset{2s}{\Longrightarrow} H$.*

We will invoke this result when we introduce unswitchable graphs later on. To prove Theorem 2, we first prove the following result, given a non-increasing degree sequence d as above.

Theorem 3 *If G be a graph on n vertices such that $d_G(v_i) = d_i$, then there exists a graph G' such that $G \overset{2s}{\Longrightarrow} G'$ with $N_{G'}(v_1) = \{v_2, \ldots, v_{d_1+1}\}$.*

Proof: Let $d = \Delta(G)(= d_1)$ be the maximum vertex degree of G. Assume there exists a v_i such that $\{v_1, v_i\} \notin E$ for i in the range $[2, d+1]$. Instead, there is an index $j \geq d+2$ such that $\{v_1, v_j\} \in E$. Again, as $j > i$, according to our assumption on the degree sequence, $d_j \leq d_i$. If V_i and V_j are the subsets of vertices of V that v_i and v_j are connected to respectively, $V_i - V_j \neq \emptyset$. Hence there exists t such that $\{v_i, v_t\} \in E$, but $\{v_j, v_t\} \notin E$. Thus we can make a 2-switch so that v_1 is adjacent to v_i. We repeat this till all the vertices adjacent to v_1 have indices in the range $[2, d+1]$. ■

Berge's theorem is easily proved by induction on the number of vertices of the graphs G and H. The condition is sufficient as $G \overset{2s}{\Longrightarrow} H$ means that the vertex degrees are preserved. Conversely, by applying Theorem 3 to each of the graphs G and H we can find a vertex v such that in graphs G' and H' respectively where $G \overset{2s}{\Longrightarrow} G'$ and $H \overset{2s}{\Longrightarrow} H'$, the neighborhood of v is identical. Now the reduced graphs $G' - v$ and $H' - v$ have the same vertex degrees and by the induction hypothesis $G' - v \overset{2s}{\Longrightarrow} H' - v$. Consequently, $G' \overset{2s}{\Longrightarrow} H'$. Combining this with the fact that $H' \overset{2s}{\Longrightarrow} H$ by a sequence of reverse 2-switches, the necessity is proved.

Here's is an interesting application of Berge's result. Consider the example below where we want to reduce graph G to graph H by 2-switches so that v_1 is adjacent to v_2 and v_3. We achieve this by switching the pair of edges $\{v_2, v_3\}$, $\{v_1, v_4\}$ with the non-existing pair of edges $\{v_1, v_3\}$, $\{v_2, v_4\}$.

Now, we can prove Theorem 1.

Proof: Consider the *if* direction. Let G be a graph on $n-1$ vertices with the degree sequence:

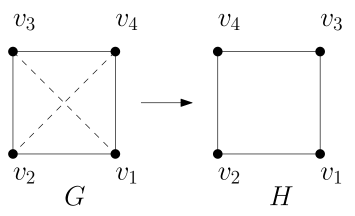


Fig. 2.2.1: Graph H obtained from G by a 2-switch

$\langle d_G(v_2) = d_2 - 1, d_G(v_3) = d_3 - 1, \ldots, d_G(v_{d_1+1}) = d_{d_1+1} - 1, d_G(v_{d_1+2}) = d_{d_1+2}, d_G(v_{d_1+3}) = d_{d_1+3}, \ldots, d_G(v_n) = d_n \rangle$

Add a new vertex v_1 and the edges $\{v_1, v_i\}$ for all $i \in [2, d_{d_1+1}]$. Then in the new graph H, $d_H(v_1) = d_1$, and $d_H(v_i) = d_i$ for all $i \geq 2$.

For the *only if* direction, assume $d_G(v_i) = d_i$. By the Lemma proved earlier and Berge's result, we can assume that $N_G(v_1) = \{v_2, \ldots, v_{d_1+1}\}$. But now the degree sequence of $G - v_1$ is as above.

Example 1 *The sequence $\langle 4, 4, 4, 3, 2, 1\rangle$ is graphical since the following sequence of reduced sequences are each graphical: $\langle 3, 3, 2, 1, 1\rangle$, $\langle 2, 1, 1, 0\rangle$ (this is obtained by a reordering of $\langle 2, 1, 0, 1\rangle$, obtained from the previous sequence), $\langle 0, 0, 0\rangle$. The last sequence corresponds to an empty graph, and the graph corresponding to the initial sequence is easily constructed.*

It should be pointed out that Hakimi's algorithm will work if the sequence element that we choose to saturate is any element of the sequence. If its degree is d_i, we reduce the d_i *highest degree elements* by 1. This observation is due to Kleitman and Wang [11].

2.3 Split graphs

A graph G is said to be a split graph if there exists a disjoint partition of its vertex set V into a complete induced subgraph on V_2 vertices and an independent (stable) set of V_1 vertices. Fig. 2.3.1 shows an example of a split graph where the induced subgraph on the vertices $\{2, 4\}$ is complete and the subset of vertices $\{1, 3\}$ form an independent set.

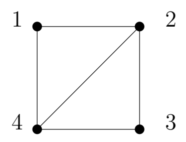


Fig. 2.3.1: A split graph

The partition of the graph into a complete graph and an independent set is not unique. For the example split graph, $\{1, 2, 4\}$ and $\{3\}$ is another partition into a (maximal) complete graph and an independent set.

There are other characterizations of split graphs. For example, this: A graph G is a split graph iff it does not contain any of the graphs in Figure 2.3.2 as induced subgraphs.

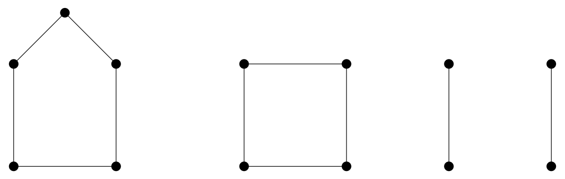


Fig. 2.3.2: Forbidden subgraphs of a split graph

There is yet another characterization of a split graph in terms of the degrees of its vertices [8]. Let $d = (d_1, d_2, \ldots, d_n)$ be the sequence of degrees of its vertices, with $n - 1 \geq d_1 \geq d_2 \geq d_3 \geq \ldots \geq d_n \geq 0$. Let m be the maximum index i for which $d_i \geq i - 1$. Call it the split index.

Then G is a split graph iff:

$$\Sigma_{i=1}^{m} d_i = m(m-1) + \Sigma_{i=m+1}^{n} d_i \tag{1}$$

Thus for the example split graph of Fig. 2.3.1, we have $d = (3, 3, 2, 2)$, $m = 3$ and both sides of Eqn.(1) evaluate to 8.

This last characterization forms the basis for an easy recognition algorithm for split graphs. Going left to right in the degree sequence, find the split index m. Verify that there is a complete graph on the first m vertices with degree d_m. Check that the remaining vertices form an independent set, with edges to the clique vertices to saturate their degrees and the residual degrees of the clique vertices.

The forbidden subgraph characterization is of interest to us. If a split graph has a 4-cycle or its complement as an induced subgraph then it is switchable. The question is: Are all split-graphs switchable ? We explore this matter in the next section.

2.4 Unswitchable graphs

A P_4 is a chordless path on 4 vertices of G, while a C_4 is a 4-cycle and a $2K_2$ (the complement of a 4-cycle) is a subgraph with 2 disjoint edges of G.

Clearly, an unswitchable graph G cannot have a P_4, a C_4 or a $2K_2$ as an induced subgraph on 4 vertices. Since no switching is possible, we cannot use 2-switches to transform a given graph G to a graph G' with the same degree sequence.

Extrapolating from the forbidden induced subgraph characterization of unswitchable graphs, Eggleton [5] proposed the following constructive charaterization of unswitchable graphs.

Theorem 4 *[5] For any positive integer n, let $\{S_i : 1 \leq i \leq 2n\}$ be a family of pairwise disjoint finite (possibly empty) sets, with union V. Let G with a vertex set V be such that any two distinct vertices $a \in S_i$ and $b \in S_j$, with $i \leq j$, are adjacent in G just if $i + n < j$ or $i > n$. Then G is unswitchable; moreover, every unswitchable graph is obtained by this construction.*

Proof: (Ours) We show that the graph constructed cannot have any of the graphs P_4, C_4 or $2K_2$ as an induced subgraph. We argue the case of C_4. Let the labels of the vertices of C_4 be a, b, c, d in cyclic order. Since a and c are not connected both cannot be in sets with indices greater than n. Let a be in a set S_i with index $i \leq n$. Since a is joined to both b and d they are in sets S_j and S_k with indices greater than $i + n$. Thus b and d must be connected. This contradicts the assumption that the induced graph on a, b, c, d is a C_4.

Similar argumemts can be made for the non-existence of $2K_2$ and P_4 as induced subgraphs.

Now for the second half of the theorem. Let G be a given unswitchable graph. It is a split graph, as follows from the forbidden subgraph characterization of split graphs. Let there be m edges connecting a vertex of the independent set with a vertex of the clique. If $\{u, v\}$ is one such edge, let $u \in S_{i_1}$ and $v \in S_{j_1}$. Then we must have $j_1 - i_1 > n$. Thus we have m such inequalities corresponding to the m edges.

Further, $j_1 > n$ and $i_1 \leq n$ for each pair of indices corresponding to the m edges. This means that we have to choose m pairs of points (indices) in the polygonal region in the $x - y$ plane, bounded by the lines $x - y > n$, $x > n$ and $y \leq n$.

We choose a minimum n such that m pairs of points can be found in this polygonal region. For the unswitchable graph of Figure 2.4.1 a distribution of its vertices among the sets S_i is shown in Figure 2.4.2 ■

Following the theorem, we constructed the unswitchable graph shown in Figure 2.4.1, setting $n = 2$.

The sets S_i, the membership of the vertices in these sets and the mutual adjacencies of the vertices are shown in Figure 2.4.2.

Here's another example, where we have gone in the opposite direction, setting $n = 2$

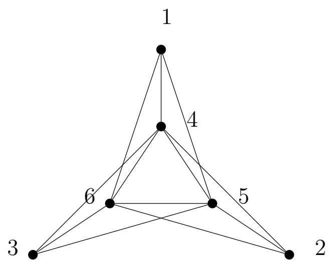


Fig. 2.4.1: A unswitchable graph G_1

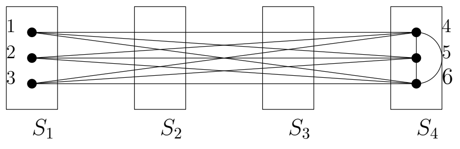


Fig. 2.4.2: Set distribution of the vertices of G_1

again and constructing the sets S_i, for $i = 1, 2, \ldots, 2n$, and adding edges between vertices in pairs of sets S_i and S_j for $i \leq j$, satisfying the other constraints on i and j.

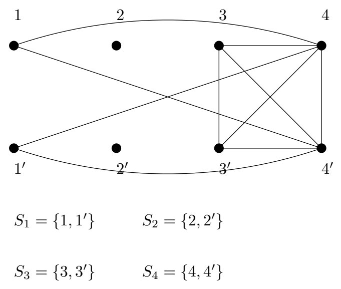


Fig. 2.4.3: Distribution of the vertices among the sets S_i

Both the graphs of Figure 2.4.1 and Figure 2.4.3 are split-graphs. This leads us to speculate on what might be the relationship between these two graph classes: split-graphs and unswitchable graphs.

It appears that the class of split graphs has an overlap with the class of unswitchable graphs. As evidence, we have the graphs of Figure 2.4.1 and Figure 2.4.3 which are split graphs but not switchable. On the other hand the graph of Figure 2.4.4 is a split graph but switchable as there exist several P_4's as induced subgraphs.

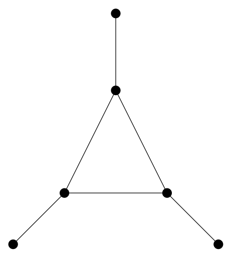


Fig. 2.4.4: A split graph that is switchable

An interesting problem is to construct a switchable graph that is not a split graph. Consider a graph G consisting of two copies of the graph of Figure 2.4.1. This graph is not a split graph but it is switchable. Indeed, by running our implementation of Hakimi's algorithm [7] on the degree sequence $d = (5, 5, 5, 5, 5, 5, 3, 3, 3, 3, 3, 3)$, we obtained the graph of Figure 2.4.5 as output. This is not a split graph as there is an induced 4-cycle on the vertex set $\{2, 3, 9, 10\}$ and is a switchable graph for the same reason.

The above considerations lead us to make the following claim.

Claim 1 *Unswitchable graphs are a proper subclass of split graphs.*

Proof: This is true since the graphs defined by Eggleton's result are all split graphs. The vertices in the sets with indices at most n constitute an independent set and the ones with indices greater than n form a complete graph. The inclusion is proper since we have found a split graph that is switchable (Figure 2.4.4). ■

In view of Claim 1, we can design a recognition algorithm for *unswitchable graphs.* Given an input graph, we first run a recognition algorithm for split graphs (for example, the degree sequence based recognition algorithm mentioned in the previous section) and if the output is true, check that the graph does not have a P_4 as an induced subgraph. For this we proceed as follows.

The recognition algorithm returns a split index m as discussed in the Section 2.3 so that the vertices with degrees $d_{m+1} \geq d_{m+2} \geq \ldots \geq d_n$ constitute an independent

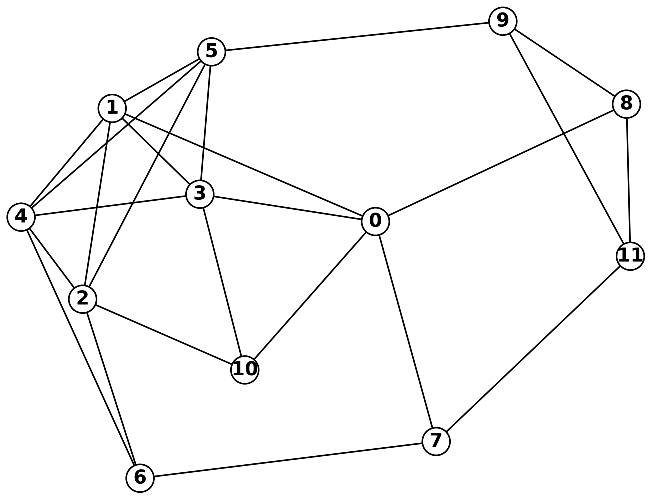


Fig. 2.4.5: A graph that is switchable but not split

set. Knowing this, from the adjacency list of the input graph, we find the adjacency list of each vertex of the independent set (Figure 2.4.6).

We use this information to construct another adjacency list that gives for each vertex of the complete graph the vertices of the independent set that are adjacent to it.

Now, for an edge $\{u, v\}$ of the clique we can find the sets of vertices S_u and S_v of the independent set that are adjacent to u and v respectively. If the set differences $S_u - S_v$ and $S_v - S_u$ are both nonempty then there exists a path P_4 betweeen u and v, making the graph switchable. If there exists no clique edge $\{u, v\}$ for which this is true then the graph is unswitchable.

Consider the graph of Figure 2.4.1 without the edges 2-6 and 3-5. The adjacency list for the vertices of the independent set and the adjacency list for the vertices of the complete graph derived from it are shown in Figure 2.4.6.

For the edge u-v = 6-5, $S_u = \{1, 3\}$ and $S_v = \{1, 2\}$. The set differences are $\{2\}$ and $\{3\}$. Since these are both non-empty, there is a P_4 path: 3-6-5-2. This shows that

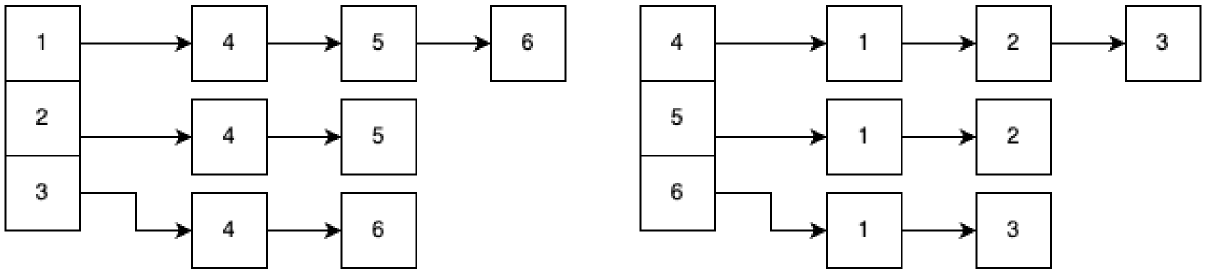


Fig. 2.4.6: Adjacency Lists for the modified Graph of Fig. 2.4.1

the modified graph is switchable.

A formal description of the recognition algorithm is given in Algorithm 2.4.1 below. The time complexity of the recognition of the algorithm is $O(n_1 n_2^2)$, where n_1 and n_2 are respectively the sizes of the independent set and the clique set. Since n_1 and n_2 are both bounded by n, $O(n^3)$ is a more succinct description of the complexity of the algorithm.

Improving the complexity of the above recognition algorithm is left as an open problem.

2.5 Generating an unswitchable graph

The second half of the proof of Eggleton's theorem requires an unswitchable graph as input. We would also like to test the recognition algorithm of the previous section on instances of unswitchable graphs. Motivated by these applications, we consider the problem of generating an unswitchable graph on n vertices by an independent method.

We first generate a split graph. Let n be the number of vertices V of the graph, obtained as (user) input. We partition V into two disjoint non-empty subsets V_1 and V_2 of size n_1 and n_2 respectively. We assume that $n_2 \geq 2$ to avoid trivial cases. Construct a complete graph on the vertices of V_2. For each of the remaining n_1 vertices of V_1, choose a random integer p in the range [0, n_2] and join the chosen vertex to a

Algorithm 2.4.1 *UnswitchableGraphRecognition*(G)

Input: The adjacency lists of the vertices of a graph G

Output: G is switchable or not

```
1: Extract the degree sequence, $d = d_1 \geq d_2 \geq \ldots \geq d_n$ of $G$
2: output ← Run the recognition algorithm for a split-graph on $d$
3: if (output = YES) then
4:     Let $m$ be the split index
5:     Extract adjacency lists of the vertices with degrees $\geq d_{m+1}$
6:     for each edge $u - v$ of the complete graph on the vertices with degrees $\leq d_m$: do
7:         Compute the neighborhoods $S_u$ and $S_v$ of the end points in the independent set
8:         Compute $S_u - S_v$ and $S_v - S_u$.
9:         if $S_u - S_v$ and $S_v - S_u$ are disjoint and non-empty: then
10:            return "G is switchable"
11:        else continue
12:        end if
13:    end for
14:    return "G is unswitchable"
15: else
16:    return "G is switchable"
17: end if
```

random subset of vertices of V_2 of size p.

We now proceed as in the algorithm for recognizing a split graph with a small change. For each pair of vertices $\{u, v\}$ in the independent set V_1, we determine the set of neighbors S_u and S_v in the set of clique vertices V_2. Compute the difference sets S_u-S_v and S_v-S_u. If these are non-empty and disjoint, for each pair of vertices x and y in the difference sets we have a P_4, defined by u-x-y-v.

A formal algorithm for generating these P_4's is described in Algorithm 2.5.1 below. If no induced subgraph isomorphic to a P_4 has been found, then we have an unswitchable graph. Othewise, we introduce new edges (chords) to eliminate the P_4's. This in turn will generate new P_4's formed by pairs of the newly introduced chords. Once again chords are introduced to eliminate the new P_4's. We continue until only one new P_4 is generated.

Algorithm 2.5.1 *findP4s*

```
1: Input: Adjacency list of graph $G$ and vertices of the independent set $V_1$
2: Output: List of all $P_4$s in $G$
3: procedure FIND_ALL_P4S($adjacency\_list, V_1$)
4:     $ListP4 \leftarrow []$
5:     for each pair of vertices $v_1$ and $v_2$ in $V_1$ do
6:         $S_1 \leftarrow$ set of neighbours of $v_1$ read from $adj\_list$
7:         $S_2 \leftarrow$ set of neighbours of $v_2$ read from $adj\_list$
8:         $S_{12} \leftarrow S_1 - S_2$
9:         $S_{21} \leftarrow S_2 - S_1$
10:        if both $S_{12}$ and $S_{21}$ are non-empty then
11:            for each pair $a, b$ where $a \in S_{12}$ and $b \in S_{21}$ do
12:                Add $[v_1, a, b, v_2]$ to $ListP4$
13:            end for
14:        end if
15:    end for
16:    return $ListP4$
17: end procedure
```

Consider the example of Figure 2.5.1, where $V_1 = \{a, b\}$ and $V_2 = \{1, 2, 3, 4\}$. Apart from the edges of the clique on V_2, we have introduced edges $\{a1\}$, and $\{b2, b3, b4\}$.

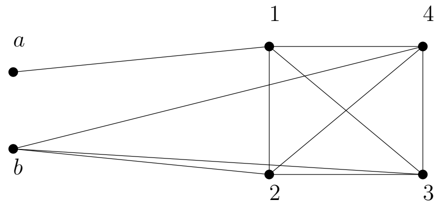


Fig. 2.5.1: First step in generating an unswitchable graph

For each one of the edges of the clique we consider the induced P_4 formed with pairs of vertices in the set $V_1 = \{a, b\}$. There are three of them as shown in Figure 2.5.2.

These induced subgraphs can be taken care of by introducing one the edges in $\{a2, b1\}$, $\{a3, b1\}$ and $\{a2, b1\}$ in the induced P_4's from left to right. All three can be taken care of by introducing the edge $b1$ in the three induced P_4's. The updated graph is shown in Figure 2.5.3, with the newly added edge as a dashed segment.

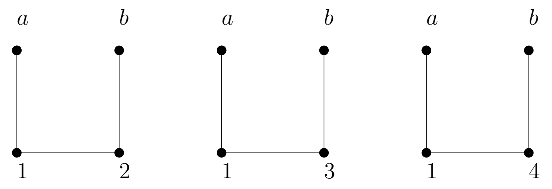


Fig. 2.5.2: Second step in generating an unswitchable graph

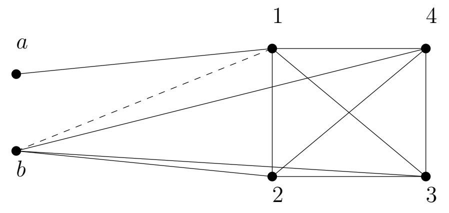


Fig. 2.5.3: Third step in generating an unswitchable graph

We immediately see that this as a problem of finding an approximate minimum cover for a class of 2-element sets in the general case. We have to go further.

Introducing these new edges can give rise to new induced P_4's. These are found by examining pairs of newly introduced edges and checking whether a P_4 is induced by these edges and the edge joining their end points in the set V_2. Once again, we generate a class of 2-element sets, for which we solve an approximate minimum cover problem. We continue this iteratively, until we reach a stage when we have a cover of size one.

In the chosen example above, the process comes to an end in one step.

We describe formally the algorithms to generate an instance of the vertex cover problem introduced at each stage and since it is an NP-complete problem a minimum-vertex degree heuristic used to add as few chords as possible to eliminate the P_4's in Algorithm 2.5.2 and Algorithm 2.5.3 below.

Algorithm 2.5.2 *vertexCoverInstanceGeneration*

```
 1: Input: $ListP4$ from $G$
 2: Output: An ordered dictionary with the two vertices that define a chord as key
    and value the frequency of occurrences of the chord.
 3: procedure FIND_ALL_2_ELEMENT_SETS($ListP4$)
 4:     $two\_element\_set\_count \leftarrow \{\}$
 5:     for each $e \in ListP4$ do
 6:         $two\_element\_inst\_one \leftarrow$ first and third element of $e$
 7:         $two\_element\_inst\_two \leftarrow$ second and fourth element of $e$
 8:         if $two\_element\_inst\_one$ is present in $two\_element\_set\_count$ then
 9:             Increment the value by 1 for key $two\_element\_inst\_one$
10:         else
11:             Set the value to 1 for key $two\_element\_inst\_one$
12:         end if
13:         if $two\_element\_inst\_two$ is present in $two\_element\_set\_count$ then
14:             Increment the value by 1 for key $two\_element\_inst\_two$
15:         else
16:             Set the value to 1 for key $two\_element\_inst\_two$
17:         end if
18:     end for
19:     Sort $two\_element\_set\_count$ in decreasing order of values
20:     return $two\_element\_set\_count$
21: end procedure
```

Algorithm 2.5.3 *edgeAddition*

```
Input: Current edge list of graph $G$, $ListP4$ and $V_1$
Output: List of new edges added to make $G$ unswitchable.
procedure ADD_MIN_EDGES($edge\_list$, $ListP4$, $V_1$)
    $edge\_list\_copy \leftarrow$ Copy current $edge\_list$
    $new\_edge\_list \leftarrow []$
    while $ListP4$ is not empty do
        $two\_element\_counts \leftarrow$ find_all_2_element_sets($ListP4$)
        $vertices\_to\_add \leftarrow$ first element of $two\_element\_counts$
        Add the extracted vertices from $vertices\_to\_add$ to $edge\_list\_copy$
        Add the extracted vertices from $vertices\_to\_add$ to $new\_edge\_list$
        $adjacency\_list \leftarrow$ Convert $edge\_list\_copy$ to adjacency list
        $ListP4 \leftarrow$ find_all_p4s($adjacency\_list$, $V_1$)
    end while
    return $new\_edge\_list$
end procedure
```

Finally, we put everything together and describe formally our algorithm for generating an unswitchable graph in Algorithm 2.5.4 below.

Now that we have discussed a method for generating unswitchable graphs, it is instructive to choose an n and construct sets $S_1, S_2, \ldots, S_n$, distributing the vertices of the graph in these sets so that the adjacencies are exactly the same as in the graph of Figure 2.5.3.

Set $n = 6$ and define the sets S_i as follows: $S_1 = \{b\}$, $S_2 = \{a\}$, $S_3 = S_4 = \{\}$, $S_5 = \{2, 3, 4\}$ and $S_6 = \{1\}$. From Eggleton's theorem the adjacencies of the vertices in these sets are as shown in Figure 2.5.4 and we have the same graph as in Figure 2.5.3.

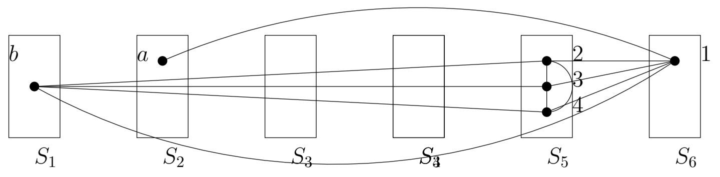


Fig. 2.5.4: Construction of the graph of Figure 2.5.3 by applying Eggleton's theorem

Algorithm 2.5.4 *unswitchableGraphGeneration*

```
1: Input: Number of vertices, $n_1$, in independent set $V_1$ and number of vertices, $n_2$, in clique set $V_2$
2: Output: An edge_list describing the graph $G$.
3: procedure GENERATE_UNSWITCHABLE_GRAPH_FOR($n_1$, $n_2$)
4:     edge_list ← []
5:     Add edges corresponding to the vertices in clique set to edge_list
6:     count_in_$V_1$ ← Pick random number of vertices in $V_1$ to connect to $V_2$
7:     while count_in_$V_1$ do
8:         vertex_$V_1$_index ← Pick a random vertex from $V_1$
9:         vertices_count_to_connect ← Pick a random number of vertices of $V_2$ to connect
10:        while vertices_count_to_connect do
11:            vertex_$V_2$_index ← Pick a random vertex in clique set
12:            edge ← (vertex_$V_1$_index, vertex_$V_2$_index)
13:            if edge not in edge_list then
14:                Add the edge to edge_list and decrement vertices_count_to_connect
15:            end if
16:        end while
17:        Decrement count_in_$V_1$
18:    end while
19:    adjacency_list ← Convert edge_list to adjacency list
20:    new_edge_list ← []
21:    ListP4 ← find_all_p4s(adjacency_list, $V_1$)
22:    if number of p4s $> 0$ then
23:        new_edge_list ← add_min_edges(edge_list, ListP4, $V_1$)
24:    end if
25:    Concatenate new_edge_list to edge_list
26:    return edge_list that corresponds to unswitchable graph $G$
27: end procedure
```

Let $N = \#P_4$ the number of P_4's discovered in the first step of the vertex cover algorithm. The time complexity of the generation algorithm is then $O(n^3 + N^2)$, where the term n^3, as in the recognition algorithm, accounts for the time complexity of identifying the P_4's and the second term is the sum obtained by adding a sequence of P_4's starting with N and decreasing to one, each term being an upper bound on the size of the vertex cover problem to be solved.

2.6 Conclusions

In this note we have proposed an algorithm for recognizing unswitchable graphs, by first showing that unswitchable graphs are a subclass of split graphs. The second half of the proof of Eggleton's theorem requires an unswitchable graph as input. Motivated by this, we have proposed an interesting algorithm for generating unswitchable graphs. The algorithms have been implemented in Python 3. In the light of Theorem 2, the degree sequence of an unswitchable graph is uniquely realizable.

A challenging open problem is to design an algorithm for generating an unswitchable graph on n vertices uniformly at random.

CHAPTER 3

Road Network Analysis of Canadian Provinces

3.1 Introduction

Motivated by the applications of the principles of complex networks to examples such as protein-protein interaction networks, social networks, etc., in this chapter, we study the statistical properties of road networks of 10 provinces of Canada. Additionally, we generate two random graphs using the Erdos Renyi model and the Configuration model respectively with the same number of nodes and edges as in the road network of each province. We compare the statistical properties both among provinces and with the random graphs generated.

3.2 Data Extraction Methodology

The OSMnx [2] Python module is a powerful toolkit for extracting, modelling, and analyzing geographical networks, with a special emphasis on road networks. OSMnx uses data from OpenStreetMap to allow users to choose a geographical location, such as a province and returns the accompanying road network as a graph. Figure 3.2.1 shows the data flow diagram (DFD) of the process that extracts a geographical network using the OSMnx library. The initial input is a province name to the function `graph_from_place(province_name)`, which outputs a directed multi-graph G(V, E). The nodes of G represent the road intersections and the edges of G represent the

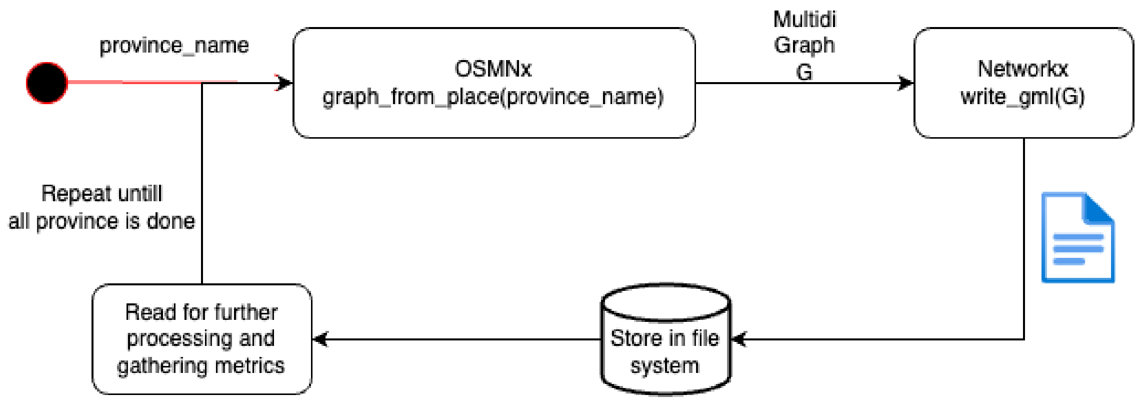

province_name
OSMNx
graph_from_place(province_name)
Multidi Graph G
Networkx
write_gml(G)
Repeat untill all province is done
Read for further processing and gathering metrics
Store in file system

Fig. 3.2.1: Flow Diagram of data extraction

road segments that define this intersection. Figure 3.2.2 shows the representation of G. G is a directed multigraph means all the edges have directions and more than one edge can connect two arbitrary nodes u, v of G. Next, using the versatile Networkx [6] python package used in the analysis of networks, graph G is saved into the file system as a GML (Graph Modelling Language) file. GML is a text-based format commonly used for representing directed or undirected graphs, making it suitable for storing complex network structures like a province's road network. This makes the task of analyzing the networks easier and also provides a way to replicate the results discussed in the subsequent sections.

In this study, the road networks of 10 provinces of Canada are extracted and analyzed. These are : Nova Scotia, Ontario, Manitoba, Quebec, Alberta, Saskatchewan, New Brunswick, British Columbia, Prince Edward Island and Newfoundland and Labrador.

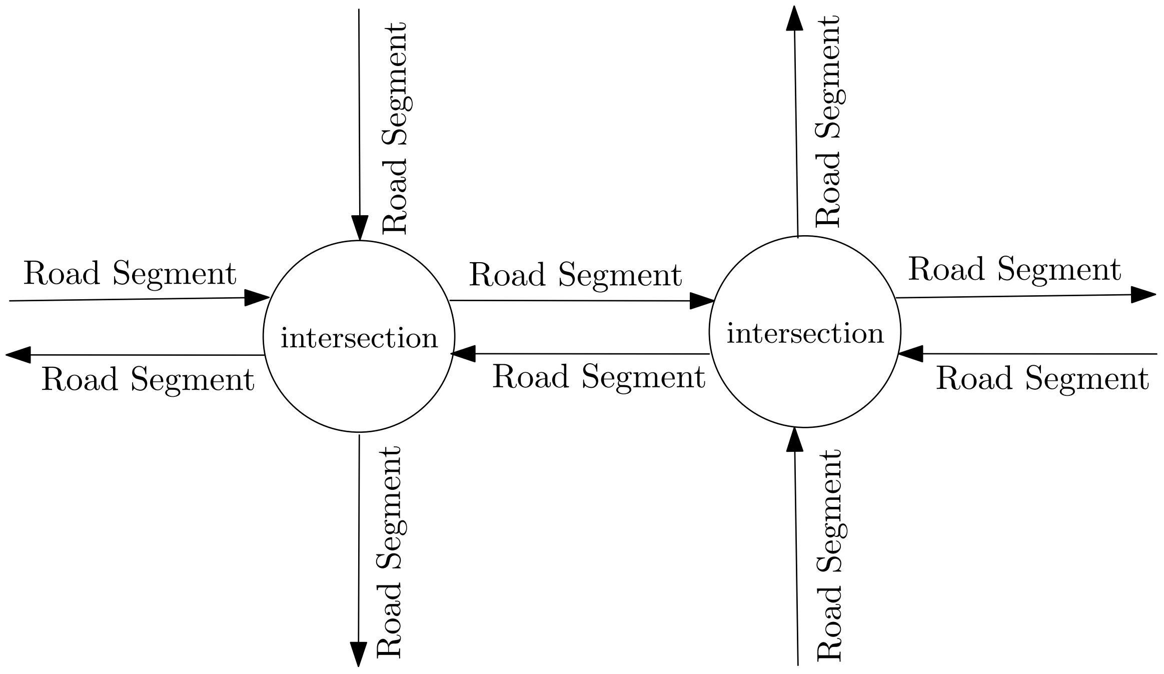


Fig. 3.2.2: Representation of graph returned from OSMNx

Province	**Number of Nodes**	**Number of Edges**
Nova Scotia	46,182	111,310
Ontario	359,535	938,322
Manitoba	70,694	197,154
Quebec	265,932	682,993
Alberta	256,831	631,772
Saskatchewan	126,405	348,263
New Brunswick	44,220	106,184
British Columbia	133,872	321,925
Prince Edward Island	10,078	25,172
Newfoundland and Labrador	35,329	80,704

Table 3.3.1: Province-wise Network Statistics

3.3 Network Statistics and Random Graph Generation

3.3.1 Network Statistics

In any network analysis, the first metric that merits scrutiny is the number of nodes and edges in the network. Table 3.3.1 encapsulates this information about the road networks of all 10 provinces. We note that there are big differences in the number of road segments and the number of intersections among the provinces. Ontario, Quebec, Alberta, British Columbia and Saskatchewan have more than 100k nodes or intersections whereas the other provinces have fewer, with Prince Edward Island having the least of them all with just over 10k intersections. The number of edges or road segments also follows a similar pattern. This is expected as the road network is built to cater to the population in these provinces and the population table 3.3.2 below validates our hypothesis. The population data of each province has been obtained from the Statscan website [16].

Province	**Number of Nodes**	**Number of Edges**	**Population**
Ontario	359535	938322	15911285
Quebec	265932	682993	8984918
British Columbia	133872	321925	5431355
Alberta	256831	631772	4800768
Saskatchewan	126405	348263	1225493
Manitoba	70694	197154	1474439
Nova Scotia	46182	111310	1069364
New Brunswick	44220	106184	846190
Newfoundland and Labrador	35329	80704	540552
Prince Edward Island	10078	25172	176162

Table 3.3.2: Province-wise Network Statistics with Population

3.3.2 Random Graph Models

A graph $G = (V, E)$ is called random if the edges joining its nodes are generated by a probabilistic mechanism. Two widely used mechanisms define respectively the $G_{n,m}$ model, also called the Gilbert model or the uniform model and the $G_{n,p}$ model, also called the Erdos-Renyi model (ER model,for short) or the binomial model.

In the first model, a graph is selected from the family of all graphs on n nodes with m edges with the uniform probability $1/((n, 2), m)$. In the second model, we choose a p such that $0 \leq p \leq 1$, and a graph on n nodes with m edges is chosen with probability $p^m(1 - p)^{(n,2)-m}$.

In a simple undirected graph m $< \binom{n}{2}$ but the road networks under discussion are directed multigraphs so this bound on the edges may not hold. With the help of iGraph [4], a Python module, a random graph with a given number of nodes and a given number of edges is generated for each province.

The configuration model is another random graph generation model. In this model, the degree sequence of the graph is given as input and a realization of a graph with the given degree sequence is chosen at random. Since the province networks are directed, to generate a random graph using the configuration model both in-degree sequence and out-degree sequence are identified and given as input to Networkx library to generate the random graph, again with the same number of vertices and the same number of edges for each province. In the upcoming sections, various network parameters are studied for each one of the actual network and compared with the same parameters of the two companion random networks.

3.4 Clustering Coefficient

In this section, two parameters, the local clustering coefficient and global clustering coefficient are studied. Local clustering coefficient quantifies the cliquishness of a node's neighbourhood [18]. The local clustering coefficient c_i is calculated using the formula as given in :

$$c_i = \frac{|\{(j,k)\}|}{k_i(k_i-1)} : j,k \in N_i, (j,k) \in E$$

where k_i is the out degree of node i and

$$N_i = j : (i,j) \in E$$

is the set of out neighbours of node i and E is set of all edges in the graph. The measure can be extrapolated to the entire graph by taking the average of local clustering coefficient of all the vertices in the graph given by:

$$c = \sum_{i \in V} \frac{c_i}{N}$$

where c_i represents local clustering coefficient of vertex i and N represents the total number of nodes in the graph. From the table 3.4.1 we observe that the average local clustering coefficient is high for each province in comparison with their respective random graph models. We also observe that the global clustering coefficient also shows a similar trend. This indicates the presence of clusters in all province road networks. The formula for calculating the global clustering coefficient as defined in [13] is given below. We also observe that both these values are very close among provinces.

$$c = \frac{3 \times \text{number of triangles}}{\text{number of connected triples}}$$

where the number of triangles refers to the count of triangles in the graph, and the number of connected triples represents the number of sets of three vertices that are all pairwise connected.

Graph Name	**Local Clustering Coefficient**	**Global Clustering Coefficient**
Nova Scotia	0.037104427	0.041856541
Erdos Renyi Nova Scotia	9.09E-05	0.000111951
Configuration Model Nova Scotia	6.61E-05	6.99E-05
Ontario	0.049957777	0.047601798
Erdos Renyi Ontario	1.44E-05	1.78E-05
Configuration Model Ontario	1.97E-05	1.95E-05
Manitoba	0.045995754	0.040694308
Erdos Renyi Manitoba	5.90E-05	7.10E-05
Configuration Model Manitoba	6.86E-05	8.08E-05
Quebec	0.044385356	0.046451164
Erdos Renyi Quebec	1.62E-05	1.80E-05
Configuration Model Quebec	2.20E-05	2.58E-05
Alberta	0.042446147	0.037262092
Erdos Renyi Alberta	1.71E-05	2.32E-05
Configuration Model Alberta	1.98E-05	2.08E-05
Saskatchewan	0.033833485	0.035661904
Erdos Renyi Saskatchewan	3.73E-05	3.75E-05
Configuration Model Saskatchewan	4.36E-05	5.36E-05
New Brunswick	0.041234287	0.046182481
Erdos Renyi New Brunswick	0.000103437	0.000105715
Configuration Model New Brunswick	0.000105102	0.000110583
British Columbia	0.04529786	0.042552878
Erdos Renyi British Columbia	4.28E-05	4.46E-05
Configuration Model British Columbia	2.86E-05	3.79E-05
Prince Edward Island	0.039926825	0.042793423
Erdos Renyi Prince Edward Island	0.000373186	0.000523249
Configuration Model Prince Edward Island	0.000771837	0.000607559
Newfoundland and Labrador	0.031565821	0.041185218
Erdos Renyi Newfoundland and Labrador	8.96E-05	0.000130307
Configuration Model Newfoundland and Labrador	0.000153051	0.000151161

Table 3.4.1: Clustering Coefficients

3.5 Average Shortest Distance

The next topological property studied is the average shortest distance between a given node and all other nodes in the graph. Average shortest distance estimates the shortest distance between any two nodes picked at random. Distance calculated here is the number of directed edges along the shortest path for any two nodes. This is calculated using the equation given below:

$$D = \sum_{s,t \in V} \frac{d(s,t)}{N(N-1)}$$

where: $d(s,t)$ is the shortest path from s to t and V is the set of all nodes in the graph and N is the total number of nodes in the graph. N*(N-1) is used to normalize the value calculated for each node. If there is no path from node u to node v then the default shortest distance is set to an integer max value of 2147483647. From the table, we observe that despite the size of the graph the average shortest distance between vertices is small for all the provinces. Another observation is that for all the provinces the random graph constructed using the Erdos Renyi model has a significantly higher value whereas the random graph generated using the configuration model has a value in the same scale as the provinces. The Erdos Renyi model's high value can be attributed to the presence of many unreachable paths.
Average shortest distance values are ranked among provinces and are shown in the table 3.5.2. From the table, we observe that provinces with greater numbers of nodes and edges have smaller average shortest distances. Prince Edward Island has the highest value. This pattern is expected because a higher number of nodes and edges are required to support the population and their needs.
A small-world network has the following two characteristics as defined in [18]:

- High clustering coefficient when compared to a random network of the same size.

- The average node-node distance is approximately equal to log(number of vertices).

Graph Name	**Avg Shortest Distance**
Nova Scotia	22.15222561
Erdos_Renyi_Nova Scotia	10341.70208
Configuration_Model_Nova Scotia	15.10254497
Ontario	4.900450389
Erdos_Renyi_Ontario	1073.49372
Configuration_Model_Ontario	4.268804188
Manitoba	44.67356109
Erdos_Renyi_Manitoba	4492.89145
Configuration_Model_Manitoba	34.36634661
Quebec	5.435196569
Erdos_Renyi_Quebec	1524.372326
Configuration_Model_Quebec	4.281091444
Alberta	16.01079995
Erdos_Renyi_Alberta	1789.856991
Configuration_Model_Alberta	13.86329372
Saskatchewan	20.55355934
Erdos_Renyi_Saskatchewan	2589.355722
Configuration_Model_Saskatchewan	6.988198076
New Brunswick	47.20932853
Erdos_Renyi_New Brunswick	11069.10252
Configuration_Model_New Brunswick	24.15828839
British Columbia	15.93452592
Erdos_Renyi_British Columbia	3651.772081
Configuration_Model_British Columbia	13.77707738
Prince Edward Island	295.8976174
Erdos_Renyi_Prince Edward Island	44953.9689
Configuration_Model_Prince Edward Island	147.9858415
Newfoundland and Labrador	30.97069985
Erdos_Renyi_Newfoundland and Labrador	15622.19806
Configuration_Model_Newfoundland and Labrador	18.92512441

Table 3.5.1: Average Shortest Distances

Province	Avg Shortest Distance
Ontario	4.900450389
Quebec	5.435196569
British Columbia	15.93452592
Alberta	16.01079995
Saskatchewan	20.55355934
Nova Scotia	22.15222561
Newfoundland and Labrador	30.97069985
Manitoba	44.67356109
New Brunswick	47.20932853
Prince Edward Island	295.8976174

Table 3.5.2: Average Shortest Distances Ranked

All the province networks satisfy the first criteria but not all provinces have an average shortest distance in the range of log(number of vertices). Specifically, the province of Prince Edward Island has a high average node-node distance.

3.6 Degree Distribution

The next parameter studied is the degree distribution of all the networks. In a directed graph there are two types of degrees for a vertex: in-degree and out-degree. In-degree is the number of the edges that come into the vertex and out-degree is the number of edges that leave the vertex. In this study, the degree of a vertex is defined as the sum of in-degree and out-degree of that vertex.

The degree distribution is the probability of finding a vertex with the given degree. It signifies the probability of occurrence of junctions with 2, 4, 6, etc. number of roads intersecting to form that junction. From Figures 3.6.1 and 3.6.2 below, we observe that the as expected the Erdos-Renyi model graphs exhibit exponential degree distribution as represented by the blue bell curve, but the same is not true for the network of a province and its corresponding network in the configuration model.

However, the latter two networks exhibit the same degree distribution. This can be attributed to the fact that the configuration model network graph is generated using the in-degree and out-degree sequence of that province. All the provinces exhibit a similar-looking degree distribution. This can be attributed to the fact that we are using junctions as nodes and the roads forming the junctions as edges.

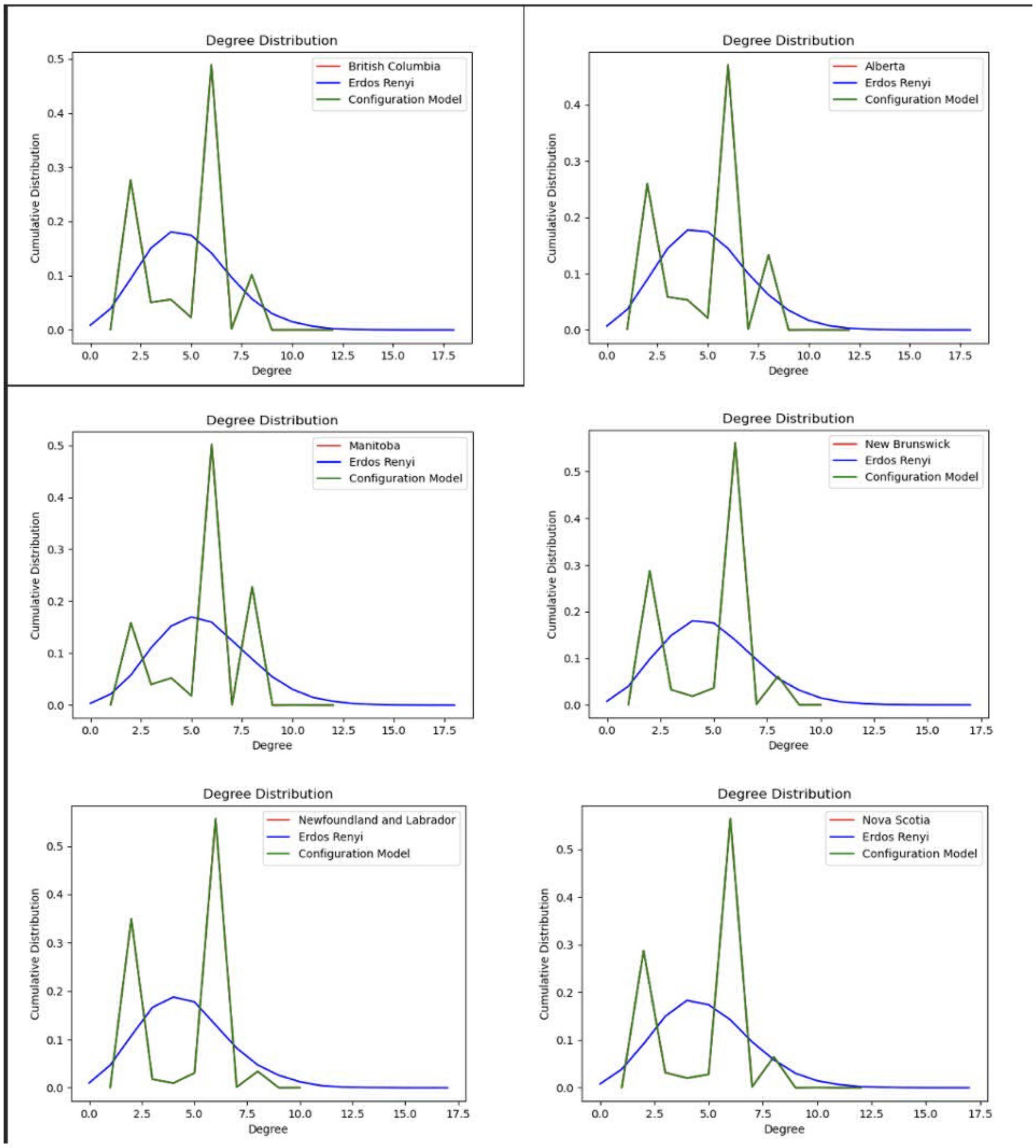


Fig. 3.6.1: Degree Distributions of Each province

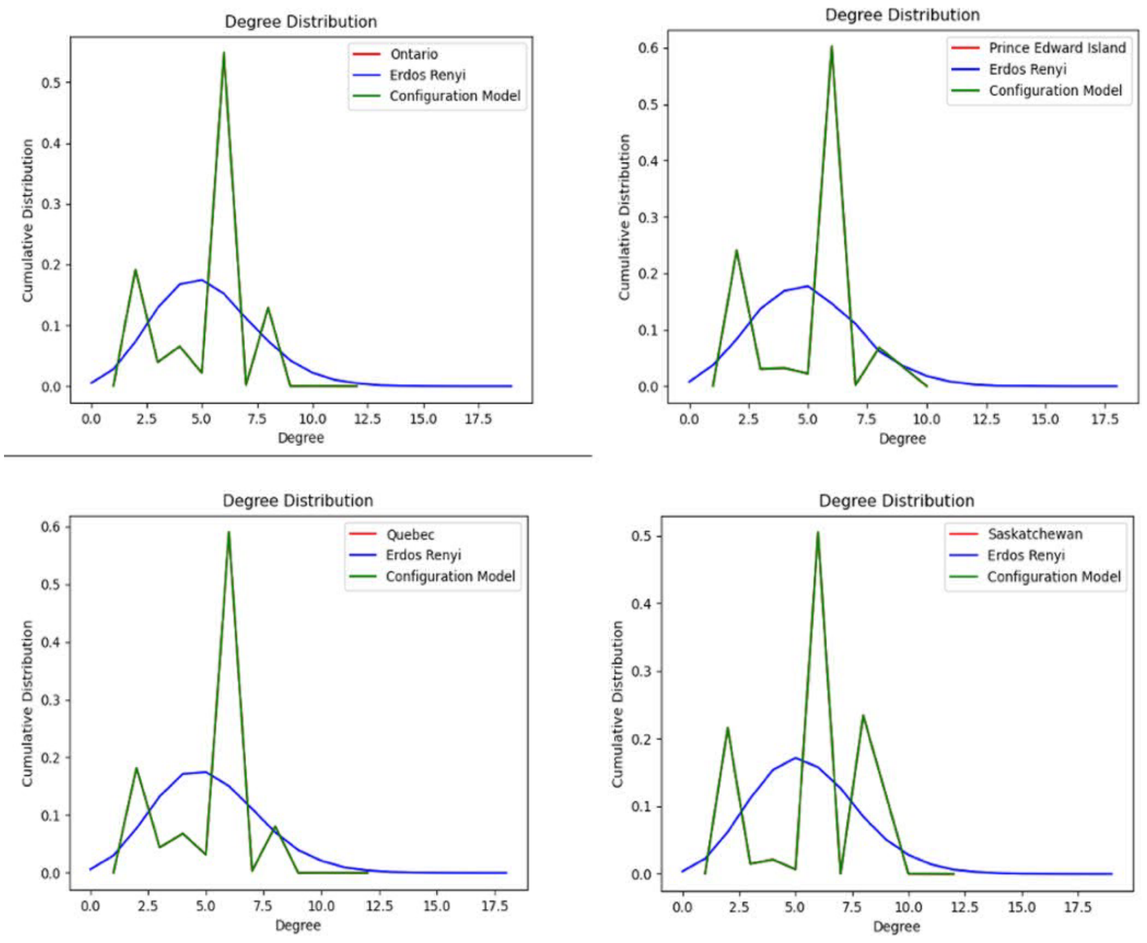


Fig. 3.6.2: Degree Distributions of Each province

3.7 Average Degree of Neighbours

The average degree of neighbours is a measure used to identify the kind of neighbours a node associates with. It is a measure to identify if there is interdependence from one intersection to another. One should expect for the actual network of the province, the average degree of neighbours increases as the degree of the node increases. This is because the purpose of the transport network is to support movement and to prevent congestion there is a low possibility that an intersection's neighbours have small degrees. From Figure 3.7.1 below, we observe that in all the province networks the average degree of nearest neighbours of a node increases with its degree, whereas for the random networks, it is relatively constant. This agrees with our hypothesis and in the next section, assortativity is also studied to validate the claim. Table 3.7.1 shows the global average degree of neighbours for all the networks.

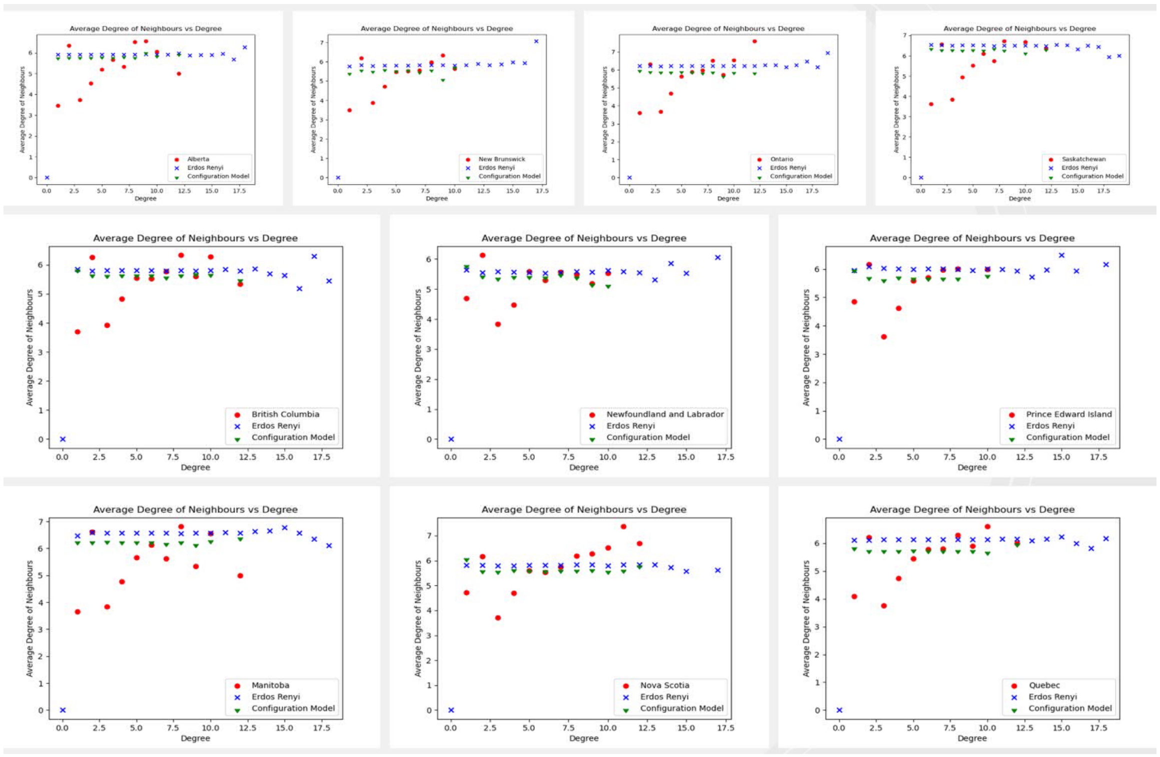


Fig. 3.7.1: Avg Degree of Neighbours

Graph Name	Avg Degree of Neighbours
Nova Scotia	5.76932931
Erdos_Renyi_Nova Scotia	5.446416528
Configuration_Model_Nova Scotia	5.684723332
Ontario	5.648767403
Erdos_Renyi_Ontario	5.959860155
Configuration_Model_Ontario	5.889503094
Manitoba	5.456051606
Erdos_Renyi_Manitoba	6.208088796
Configuration_Model_Manitoba	6.273989996
Quebec	5.507781483
Erdos_Renyi_Quebec	5.792333836
Configuration_Model_Quebec	5.78562714
Alberta	5.309459965
Erdos_Renyi_Alberta	5.616876808
Configuration_Model_Alberta	5.846647238
Saskatchewan	5.60740285
Erdos_Renyi_Saskatchewan	6.117146874
Configuration_Model_Saskatchewan	6.310307503
New Brunswick	5.274978991
Erdos_Renyi_New Brunswick	5.580254002
Configuration_Model_New Brunswick	5.528243765
British Columbia	5.373841507
Erdos_Renyi_British Columbia	5.46544186
Configuration_Model_British Columbia	5.670169289
Prince Edward Island	5.394721234
Erdos_Renyi_Prince Edward Island	5.685891107
Configuration_Model_Prince Edward Island	5.751521122
Newfoundland and Labrador	5.18232645
Erdos_Renyi_Newfoundland and Labrador	5.275580474
Configuration_Model_Newfoundland and Labrador	5.422694977

Table 3.7.1: Average Degree of Neighbors by Graph Type and Province

3.8 Topological Assortativity

In the previous section, we observed that there is a positive correlation between the degree of a node and the average degree of its neighbours. Another parameter that estimates this correlation is Assortativity. Assortativity measures to which kind of neighbours a node prefers to attach. In other words, it tries to answer the question of whether high-degree nodes prefer to attach to other high-degree nodes or not [13]. The assortativity coefficient (r) is calculated using:

$$r = \frac{\sum_{i=1}^{M}(j_i - \mu_j)(k_i - \mu_k)}{\sqrt{\sum_{i=1}^{M}(j_i - \mu_j)^2 \sum_{i=1}^{M}(k_i - \mu_k)^2}}$$

where:

- j_i and k_i are the degrees of the nodes at the ends of the i-th edge.
- μ_j and μ_k are the mean degrees of the nodes at the ends of all edges.
- M is the total number of edges in the network.

Value of r falls in the (-1,1) with 1 representing high assortative mixing meaning high-degree nodes prefer to attach to other high-degree nodes, 0 representing random connection that is there is no preference and -1 represents high disassortative mixing meaning high-degree node prefers to attach to low degree nodes. For a directed graph, the above formula has two possibilities in-assortativity and out-assortativity. For in-assortativity, the in-degrees of the nodes are used and for out-assortativity, the out-degrees are used in place of degrees mentioned in the formula. Both in-assortativity and out-assortativity are shown in table 3.8.1 below.

From the table 3.8.1, we observe that the both in-assortativity and out-assortativity coefficients are relatively high and are positive for all province networks barring the road network of Newfoundland and Labrador. This implies that there is assortative mixing in these networks except for the province of Newfoundland & Labrador and is consistent with the results from the graphs shown above.

Graph Name	In Assortativity	Out Assortativity
Nova Scotia	0.068186012	0.066716534
Erdos_Renyi_Nova Scotia	0.002290793	0.000743063
Configuration_Model_Nova Scotia	0.000123939	-0.00030633
Ontario	0.19551312	0.195076679
Erdos_Renyi_Ontario	-2.41E-05	-8.83E-06
Configuration_Model_Ontario	4.81E-06	0.000265914
Manitoba	0.227503019	0.228605825
Erdos_Renyi_Manitoba	0.000910535	-0.000926127
Configuration_Model_Manitoba	0.001480432	0.001776707
Quebec	0.171609501	0.172168904
Erdos_Renyi_Quebec	-0.000200242	0.000176849
Configuration_Model_Quebec	0.000283334	0.000444947
Alberta	0.163011864	0.174027829
Erdos_Renyi_Alberta	-0.000486747	-0.000112093
Configuration_Model_Alberta	0.000525548	0.000447223
Saskatchewan	0.148661789	0.149469205
Erdos_Renyi_Saskatchewan	0.001294267	-0.000292297
Configuration_Model_Saskatchewan	-0.001011431	-0.000948201
New Brunswick	0.043693798	0.043082098
Erdos_Renyi_New Brunswick	-0.001028574	-0.001346213
Configuration_Model_New Brunswick	0.002650523	0.002452597
British Columbia	0.130744531	0.133876507
Erdos_Renyi_British Columbia	-0.000246289	0.000522609
Configuration_Model_British Columbia	0.002160879	0.002498941
Prince Edward Island	0.061387993	0.060756557
Erdos_Renyi_Prince Edward Island	0.001641834	-0.000504828
Configuration_Model_Prince Edward Island	0.004407728	0.008131692
Newfoundland and Labrador	-0.069211687	-0.067428612
Erdos_Renyi_Newfoundland and Labrador	-0.000323387	-0.000874421
Configuration_Model_Newfoundland and Labrador	-0.008157141	-0.006541526

Table 3.8.1: Assortativity

3.9 Network Resilience

In the context of complex networks, network resilience involves studying how various types of networks, such as social networks, transportation systems, power grids, or computer networks, respond and adapt to disruptions or failures. Simulating attacks on a network can also be used to measure a network's resilience.

There are two types of network attack possible: one is a vertex-based attack and the other is an edge-based attack. Holme et al. [10] compared both types of attack for two real-world networks and four model networks, one of which is the Erdos Renyi model. Here we conduct a similar kind of study with an actual network and a corresponding random model network, generated using the configuration model.

The description of the various kinds of attacks and the inferences we can draw from these simulations are addressed in the following sections.

3.9.1 Vertex Based Attack

A vertex-based attack identifies how the network behaves when vertices are removed from a network. When we remove a vertex v all the incident edges are also removed.

There are many strategies available to remove vertices from a network. In this study, we discuss two of them: random removal of vertices and removal of vertices in descending order of vertex betweenness.

The betweenness centrality, $C_B(v)$, of a vertex v is defined as below (see [3] for more details):

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1}$$

where:

- σ_{st} is the total number of shortest paths from node s to node t.
- $\sigma_{st}(v)$ is the number of those paths that pass through the node v.
- The sum is taken over all pairs of nodes s and t, where $s \neq t$ and $s, t \neq v$.

The vertex betweenness measure identifies important vertices in the network.

In random vertex removal, we repeatedly remove 10% of the current vertices, selected at random from the network and to quantify network behaviour, the average shortest distance is then re-calculated for the residual (90%) vertices.

Another strategy we have explored is to remove repeatedly the top 10% of the current set of vertices in decreasing order of vertex betweenness and recalculate the average shortest distance of the residual (90%) vertices.

Both strategies are repeated for the random graph generated using the configuration model.

We have plotted a graph that shows how the average shortest distance varies for each province in the targetted removal of vertices (3.9.1). From this figure, we can see that for Prince Edward Island there is a steep incline in the average shortest distance as vertices are removed. For the other provinces, the incline is gradual meaning other provinces are more resilient towards targetted vertex-based attack. Another insight we can gather from this plot is that bigger networks are more resilient. Understandably so as in the bigger networks, there are multiple ways to get to a target node.

Figures 3.9.2 and 3.9.3 compare the impact of the average shortest distance on the random removal of vertices and targeted removal of vertices for the province network and the random model. Based on the definition of vertex betweenness one should expect the average shortest distance to increase at first because important vertices are removed and then merge the trend of random removal as more and more vertices

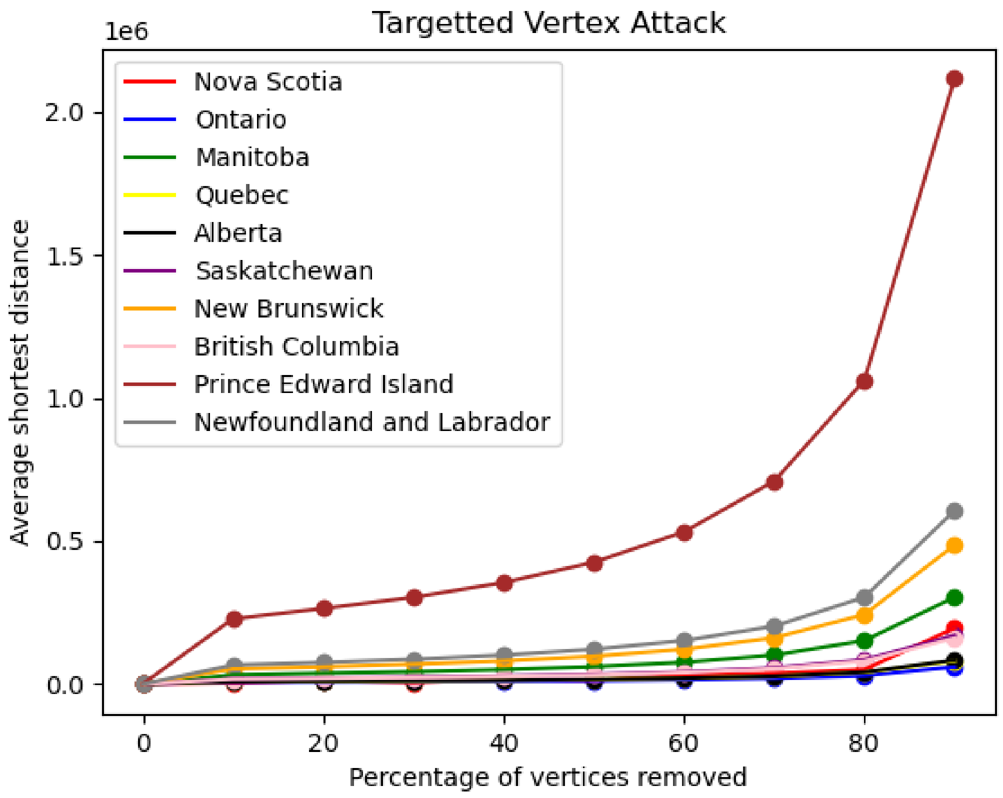


Fig. 3.9.1: Avg Shortest Distance vs Percentage of Nodes Removed

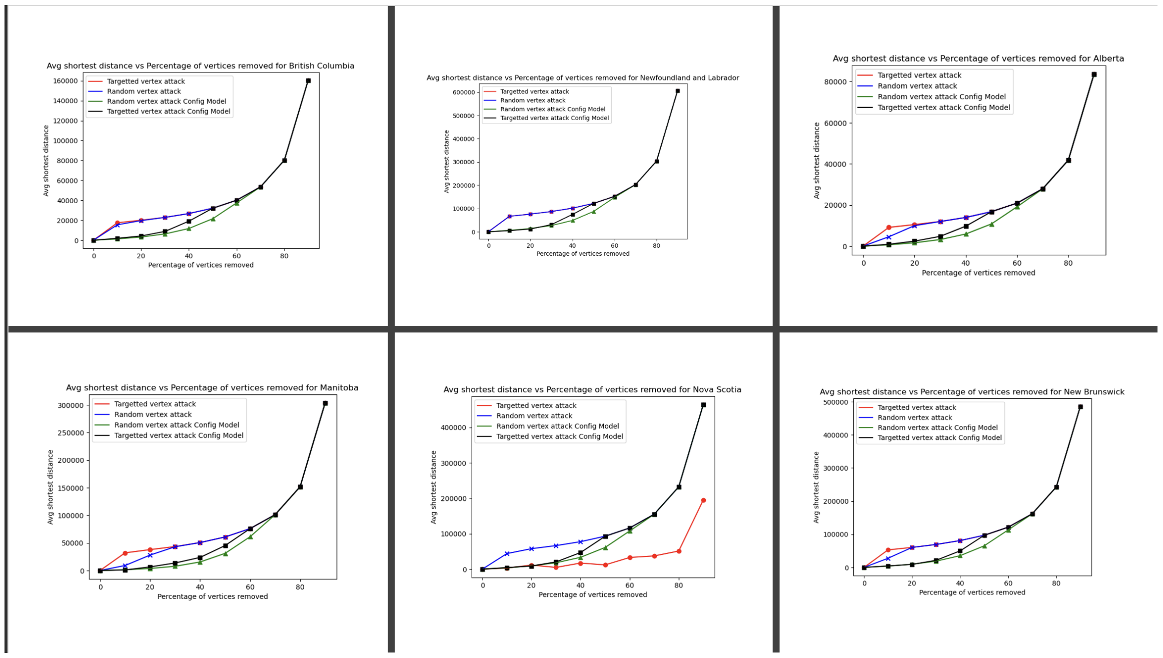


Fig. 3.9.2: Random vs Targetted Vertex Attack

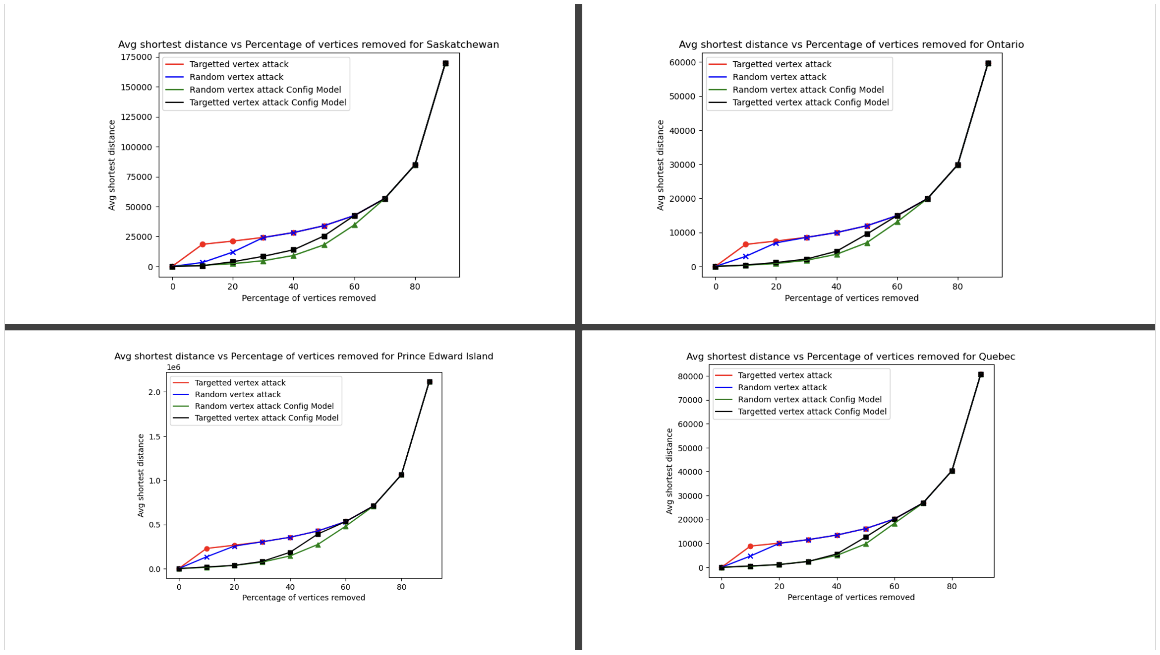


Fig. 3.9.3: Random vs Targetted Vertex Attack

are removed. This behaviour is seen in almost all provinces except for Nova Scotia. Another observation is that in the random model, the impact of vertex removal is much more gradual when compared with the actual network. One possible explanation for this could be that in the actual network, the intersection and the road segments forming the intersection are interdependent. In contrast, in the random model, it is completely probabilistic.

3.9.2 Edge Based Attack

An edge-based attack quantifies how the network behaves with the removal of edges from the network. Since the removal of an edge does not affect the vertices, repeated removal of edges leaves the network with isolated vertices.

As for vertex removals, two strategies for removing edges are explored here: one is the random removal of edges, where in every iteration 10% of edges are selected at random and removed and is repeated until 90% of the edges are removed. In every iteration, the new average shortest distance between vertices of the network is calculated and the graph of how it varies is plotted. Second is the targeted removal of edges where again 10% of the edges are removed in each iteration in the decreasing order of edge betweenness. Edge betweenness, $C_B(e)$, is calculated using the following formula [3]:

$$C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{2}$$

where:

- $C_B(e)$ represents the betweenness centrality of an edge e.
- σ_{st} is the total number of shortest paths from node s to node t.
- $\sigma_{st}(e)$ is the number of those paths that pass through the edge e.
- The summation is taken over all pairs of nodes s and t in the graph, where $s \neq t$.

Edge attacks as a whole have the same effect on province and random networks as vertex attacks seen in the previous section. From figure 3.9.4 we see that bigger networks are less susceptible to this kind of attack. Figures 3.9.5 and 3.9.6 also show that the average shortest distance increases gradually as compared with their respective actual province network. This strongly indicates that there is an interdependence between the vertices and edges of the network.

3.10 Conclusion

In this chapter, the province networks of 10 provinces are extracted. Two random graphs, one using the Erdos Renyi model and the other using the configuration model with the same number of nodes and the same number of edges as the province network are generated. Parameters like clustering coefficient, degree distribution, average degree of neighbours and assortativity are compared among provinces and with their respective random graphs. An approach to quantify network resilience is discussed

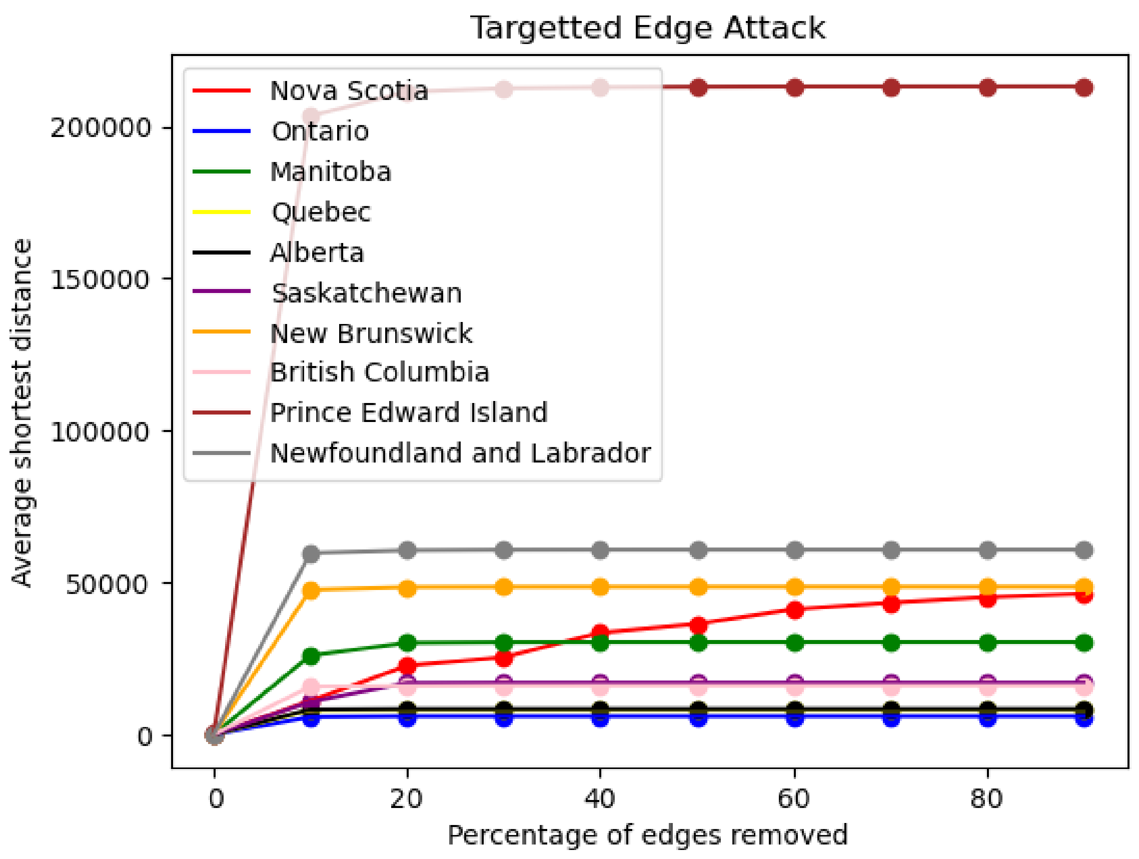


Fig. 3.9.4: Avg Shortest Distance vs Percentage of edges Removed

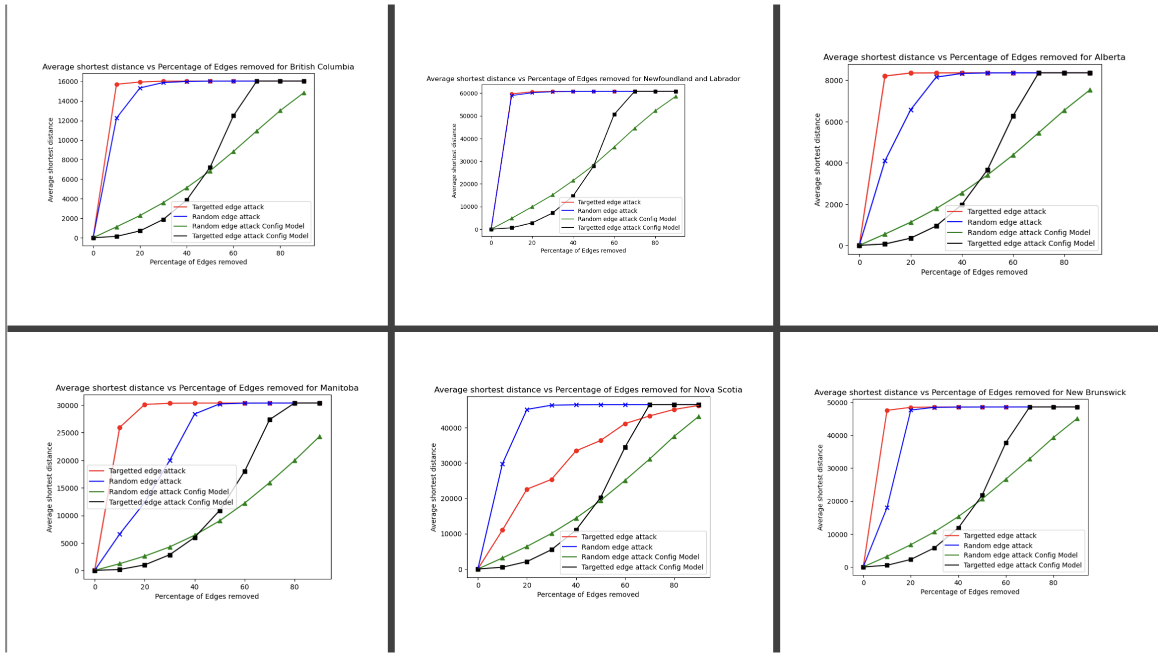


Fig. 3.9.5: Random vs Targetted Edge Attack

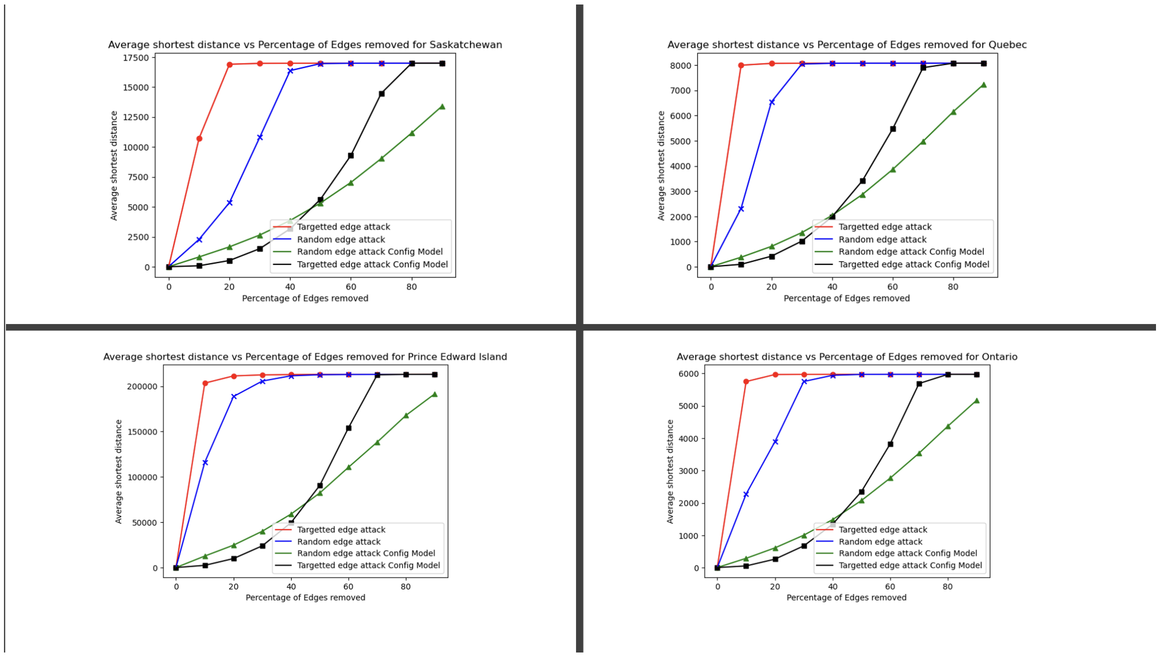


Fig. 3.9.6: Random vs Targetted Edge Attack

and is compared with one of the random models and among provinces. From this chapter, we found that the province network did not exhibit small-world properties, the larger the size of the province network more resilient it was to both vertex-based attacks and edge-based attacks and finally as expected there is an interdependence between edges and nodes of a network. From the section, we also see that the configuration model was able to model the network more closely than the Erdos-Renyi model.

CHAPTER 4

Conclusion and Future Work

In this thesis, a theoretical problem and a practical problem in the field of complex networks are explored. We begin by exploring edge switiching as a mechanism to move from one realization of a graph to another without changing the degree sequence of the graph. Edge switching was a mechanism that was explored for generating a graph uniformly at random given a graphical degree sequence but we encountered the class of unswitchable graphs where edge switching is not possible

In chapter two the class of unswitchable graphs is explored in depth. First, we prove that unswitchable graphs are a proper subclass of split graphs. Then two algorithms one to recognize whether a graph is an unswitchable graph or not given a realization of the graph and the other to generate an unswtichable graph given the number of nodes required are discussed and are presented as theoretical contributions to the field of complex networks.

In chapter three transportation network of Canada is explored. First, a method to extract the road networks of 10 provinces is discussed. Then two companion random graphs with the same number of nodes and edges as the province network are generated, one using the Erdos Renyi model and the other using the Configuration model. Complex network parameters like clustering coefficients, degree distribution, average degree of neighbours and assortativity are compared with companion random networks and among province networks. A method to quantify network resilience is discussed and implemented. The method identifies random networks being probabilistic breaks more gradually than actual networks and among provinces networks the networks with sparse connectivity break much faster than a network with more dense connectivity.

4.1 Future Works

In Chapter 3, models to generate a directed random graph are discussed. In the appendix, a method to generate a random directed graph given the in-degree sequence and out-degree sequence is discussed. The in-degree and out-degree sequences are the marginals (row sums and column sums) of the graph represented as an adjacency matrix. This method used the Markov Chain Monte Carlo (MC-MC) method to generate a random directed graph uniformly at random with the same marginals. This thesis can be extended using a random graph generated using this provided this method can scale to the size of the networks used here.

REFERENCES

[1] Berge, C. (1973). *Graphs and Hypergraphs.* North-Holland.

[2] Boeing, G. (2017). Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, environment and urban systems*, 65:126–139.

[3] Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177.

[4] Csardi, M. G. (2013). Package 'igraph'. *Last accessed*, 3(09):2013.

[5] Eggleton, R. B. (1975). Graphic sequences and graphic polynomials: a report. In *Colloq. Math. Soc. J. Bolyai*, volume 10, pages 385–392.

[6] Hagberg, A. and Conway, D. (2020). Networkx: Network analysis with python. *URL: https://networkx. github. io.*

[7] Hakimi, S. (1962). On the realizability of a set of integers as degrees of the vertices of a graph. *SIAM J. Appl. Math.*, 10:496–506.

[8] Hammer, P. L. and Simeone, B. (1981). The splittance of a graph. *Combinatorica*, 1:275–284.

[9] Havel, V. (1955). A remark on the existence of finite graphs (Czech.). *Časopis Pěst. Mat.*, 80:477–480.

[10] Holme, P., Kim, B. J., Yoon, C. N., and Han, S. K. (2002). Attack vulnerability of complex networks. *Physical review E*, 65(5):056–109.

[11] Kleitman, D. J. and Wang, D. L. (1973). Algorithms for constructing graphs and digraphs with given valences and factors. *Discret. Math.*, 6(1):79–88.

[12] Lowcay, C., Marsland, S., and McCartin, C. (2013). Constrained switching in graphs: a constructive proof. In *2013 International Conference on Signal-Image Technology & Internet-Based Systems*, pages 599–604. IEEE.

[13] Newman, M. E. (2003). The structure and function of complex networks. *SIAM review*, 45(2):167–256.

[14] Oxford Dictionaries (2017). Markov chain — definition of markov chain in us english by oxford dictionaries. Oxford Dictionaries. Archived from the original on December 15, 2017. Retrieved 2017-12-14.

[15] Rao, A. R., Jana, R., and Bandyopadhyay, S. (1996). A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 225–242.

[16] Statistics Canada (2024). Population estimates, quarterly. DOI. Retrieved from `https://doi.org/10.25318/1710000901-eng`.

[17] Taylor, R. (2006). Contrained switchings in graphs. In *Combinatorial Mathematics VIII: Proceedings of the Eighth Australian Conference on Combinatorial Mathematics Held at Deakin University, Geelong, Australia, August 25–29, 1980*, pages 314–336. Springer.

[18] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *nature*, 393(6684):440–442.

APPENDIX A

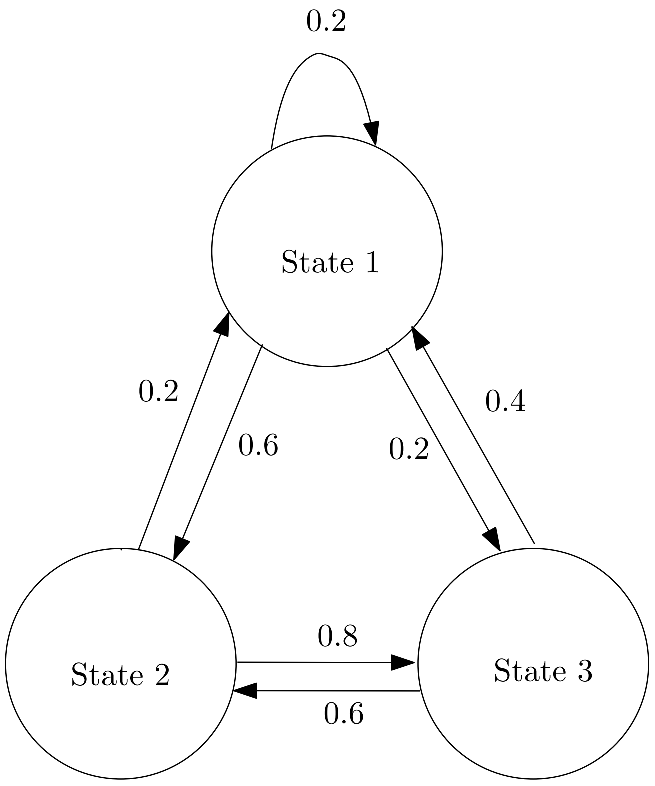
Generating a directed graph with given marginals uniformly at random

A.1 Introduction

In chapter three random models the Erdos-Renyi model and the Configuration model, were used to compare the metrics of the actual network. This served as a motivation for exploring other ways of generating a uniformly random directed graph. In Chapter 1 it is shown that a directed graph can represented as an adjacency matrix. Marginals are the row sums and column sums of a matrix. In this appendix, we explore the generation of a directed graph with the same marginals as the initial graph uniformly at random using the Markov Chain Monte Carlo (MC-MC) method given an initial directed graph.

A.2 Background

MC-MC method is a statistical method made up of two parts, one is a Markov chain and the other is Monte Carlo method. A Markov chain is a model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [14]. Figure A.2.1 shows an example of a Markov chain. The matrix in the figure A.2.1 represents the transition probability which is the probability of the system moving from the current state to the next state.



	State 1	State 2	State 3
State 1	0.2	0.6	0.2
State 2	0.2	0	0.8
State 3	0.4	0.6	0

Fig. A.2.1: Markov chain and its transition probabilities

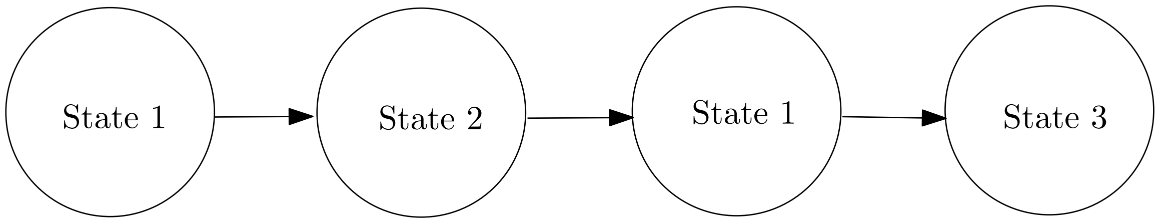


Fig. A.2.2: example of a random walk

The Monte Carlo method is a computational algorithm that relies on repeated random sampling to obtain numerical results. Here the Monte Carlo method is used to estimate the probability distribution of the Markov chain. For example, consider a random walk represented in figure A.2.2. The probabilities of each state in the chain are calculated:

P(state 1) = 2/4, P(state 2) = 1/4 and P(state 3) = 1/4

On repetition, the Markov chain converges to a stationary state. The stationary state of a Markov chain is the state at which the probability of a state remains unchanged in the Markov chain as the system moves from one state to another. Let P represent the transition probability matrix and π*P represents the probability distribution after one step in the Markov chain. Then, the stationary state of the system can be expressed mathematically as:

$$\pi = \pi * P \tag{1}$$

In the stationary state probability of each state represents the target distribution. However, not all Markov chains converge to a stationary state. Only Markov chains that satisfy the balance condition converge to a stationary state. The balance condition is given below:

$$\pi(x)P(x \to y) = \pi(y)P(y \to x) \tag{2}$$

where x, y are the states and P(x $\to$ y) represents the transition probability to move from state x to state y.

Rao et all [15] generated a random 0-1 matrix using the MC-MC method. In the next section, the algorithm presented in the paper is discussed in depth.

A.3 Rao et al's Algorithm

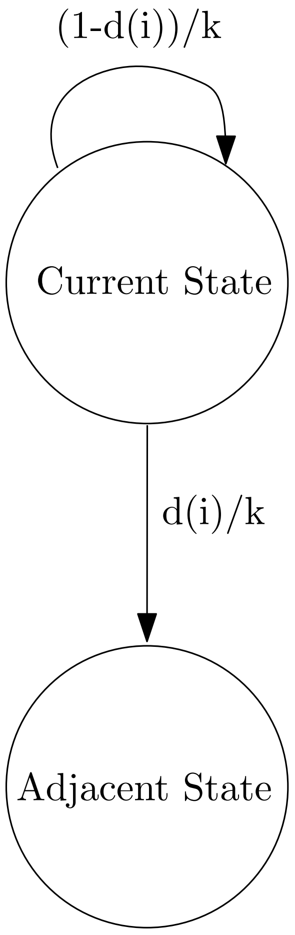
Given the marginals, two types of 0-1 matrices are possible. One is a $m \times n$ matrix with no structural zeroes and the other is a square matrix with structural zeroes. Figures A.3.1 and A.3.2 show the same. The main focus is on type 2 matrices because the adjacency matrix of the class of directed under consideration will be of type 2 as discussed previously. The main idea explained in Rao et al's paper is if there is a mechanism to move from one 0-1 matrix to another while preserving the marginals then the problem can be formulated as a Markov chain as shown in Figure A.3.3 k in figure A.3.3 is the total number of 0-1 matrices with the given marginal, in this total number of directed graphs with the same marginals as the initial graph and two

0	0	1	0
1	0	0	1
1	0	0	1
0	1	1	0

Fig. A.3.1: Example of a type 1 matrix

X	0	1	0
1	X	0	1
1	0	X	1
0	1	1	X

Fig. A.3.2: Example of a type 2 matrix



	Current State	Adj State
Current State	(1-d(i))/k	d(i)/k
Adj State	0	0

Fig. A.3.3: Problem formulated as Markov chain with its transition matrix

matrices are adjacent if one can be obtained from another on switching one alternate cycle. d(i) is the number of adjacent states available for the matrix. In other words, it is the number of alternating cycles of the matrix at the given state.

An alternate cycle can be of two types one an alternate rectangle and the other a compact hexagon

Definition 2 *An alternate Rectangle is a set of four distinct cells with indices* i_1j_1, i_1j_2, i_2j_2, i_2j_1 *with alternating zeroes and ones*

Definition 3 *Compact Hexagon is a set of 6 distinct cells with indices* i_1j_2, i_1j_3, i_2j_3, i_2j_1, i_3j_1, i_3j_2 *with alternating zeroes and ones*

Figures A.3.4 and A.3.5 show an example for each type of alternating cycle. Switching

0	0	1	0
1	0	0	1
1	0	0	1
0	1	1	0

Fig. A.3.4: Example of an alternating rectangle

X	1	0
0	X	1
1	0	X

Fig. A.3.5: Example of a compact hexagon

along an alternating cycle generates a new 0-1 matrix and preserves the marginals of the matrix. The paper discusses how only an alternate rectangle is not enough to move to all neighbouring states for the type 2 matrix. Further, the authors also provide a mathematical proof for this method.

However, estimating k is not an easy problem and it is the most important step to generate a directed graph with given marginals uniformly at random. Suppose k is very large then remaining in the current state probability is close to one and convergence to stationary state takes a long time. 3 methods for estimating k are discussed in the paper:

- Estimating k using a pilot study
- Using the maximum number of alternating cycles visited so far
- Combination of 1 and 2 where a pilot study to estimate upper bound k and also update on identifying new upper bound.

A Fortran-like pseudocode and the proofs are extensively discussed in the paper. In the next section, the flow diagram and the implementation of pseudocode are discussed.

A.4 Algorithm Flow and Implementation Details

Figure A.4.1 shows the flow diagram of the Fotran-like pseudo code given in the paper. The algorithm has two parts, part one is the pilot run to get the estimated upper bound of k and part two is moving from one 0-1 matrix to another by using an alternating cycle $3 * t$ times where t is the minimum of either number of 1's or the number of non-structural zeroes. The flow diagram is explained in detail as follows:

- Step 1. Given a starting directed graph, its adjacency matrix is used as the initial 0-1 matrix A and the set number of random graphs required to be generated in S
- Step 2. Setting up the initial variable values. For example, t is either set to minimum 0f number of 1's or the number of non-structural zeroes. Other notable variables are:

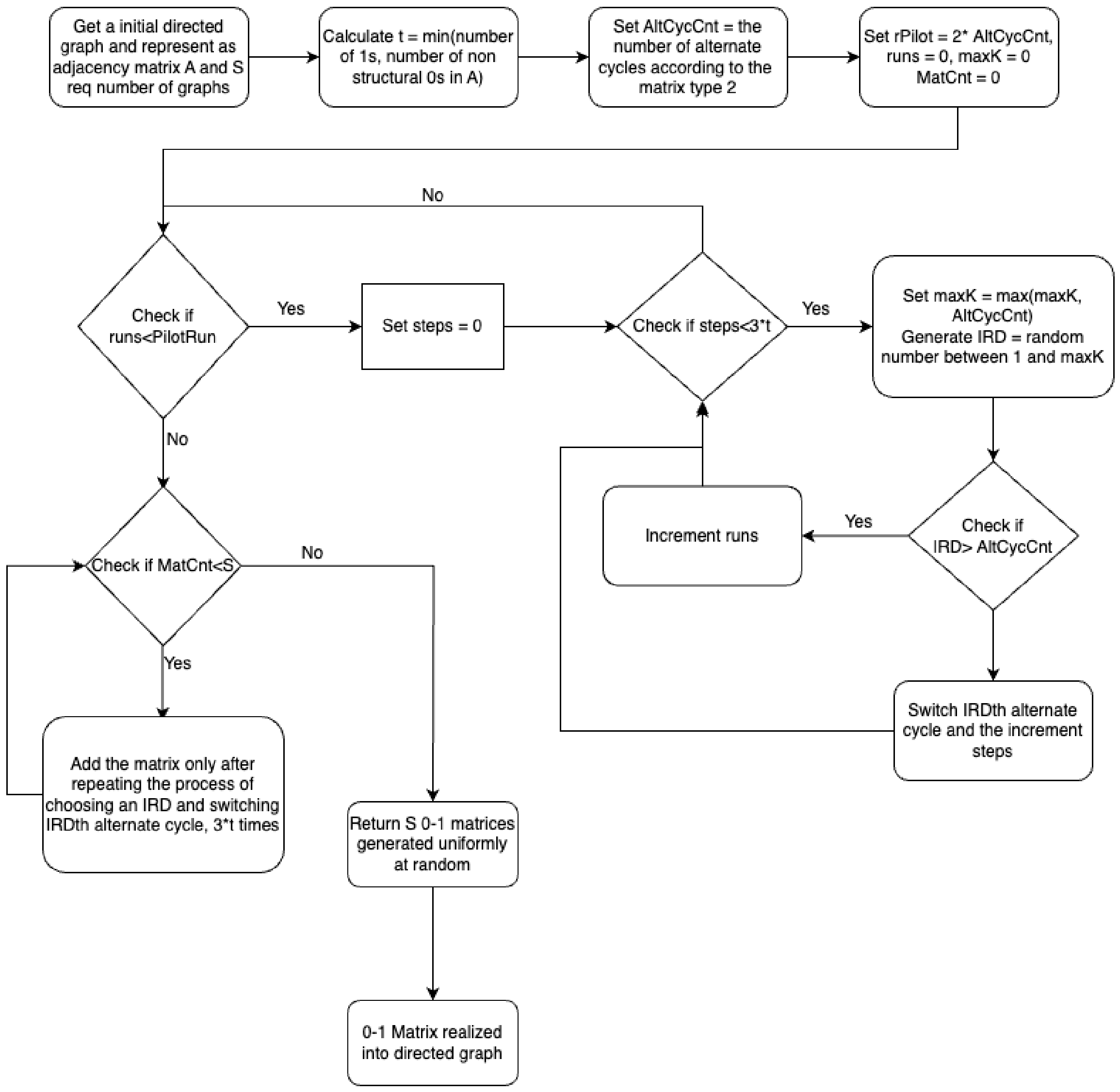

Get a initial directed graph and represent as adjacency matrix A and S req number of graphs
Calculate t = min(number of 1s, number of non structural 0s in A)
Set AltCycCnt = the number of alternate cycles according to the matrix type 2
Set rPilot = 2* AltCycCnt, runs = 0, maxK = 0 MatCnt = 0
No
Check if runs<PilotRun
Yes
Set steps = 0
Check if steps<3*t
Yes
Set maxK = max(maxK, AltCycCnt) Generate IRD = random number between 1 and maxK
No
Check if IRD> AltCycCnt
Yes
Increment runs
Switch IRDth alternate cycle and the increment steps
Check if MatCnt<S
No
Yes
Add the matrix only after repeating the process of choosing an IRD and switching IRDth alternate cycle, 3*t times
Return S 0-1 matrices generated uniformly at random
0-1 Matrix realized into directed graph

Fig. A.4.1: Flow Diagram of generating a directed graph using MC-MC method

  - AltCycCnt - number of alternating cycles for the matrix A
  - PilotRun - number of runs required to estimate upperbound maxK
  - MatCnt - matrix count to track the number of adjacency matrix got
  - steps - to track the number of times the adjacency matrix has been changed along an alternating cycle
  - maxK - is the maximum of number of alternate cycles for a matrix encountered so far

- Step 3 is performing the pilot run PilotRun times to estimate maxK
- Step 4 is performing the switching along alternating cycle 3*t times and then adding the final matrix to the list of matrix
- Step 5 Repeat until the required number of matrices is got.

The matrices got correspond to directed graphs with given marginals generated uniformly at random. The above algorithm was implemented in Python 3.

A.5 Conclusion

In this chapter, with the help of the MC-MC method a directed graph with given marginals is generated uniformly at random. This method needs more exploration on how it can scale to generate random graphs with sizes discussed in Chapter 3.

VITA AUCTORIS

NAME:	Srivatsan Vasudevan
PLACE OF BIRTH:	Chennai
YEAR OF BIRTH:	1997
EDUCATION:	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2024