PROPERTIES OF NODE RELIABILITY POLYNOMIALS

Properties of node reliability polynomials

by

Hanshu Yu

to obtain the degree of Master of Science
in Electrical Engineering
Track Wireless Communication and Sensing
at the Delft University of Technology,
to be defended publicly on Friday February 25, 2021 at 09:30 AM.

Student number:	4907787
Project duration:	March, 2021 – February, 2022
Thesis committee:	Prof.dr.ir. Rob Kooij, TU Delft, supervisor
	Dr. Huijuan Wang, TU Delft

To Boyi and my parents

PREFACE

With this thesis project, *Properties of Node Reliability Polynomial*, I finished the Master of Science degree in Electrical Engineering at the Delft University of Technology. This project has been carried out at the Network Architectures and Services (NAS) group.

Firstly, I would like to express huge gratitude to my supervisor Professor Rob Kooij for his support, encouragement, and guidance during the entire thesis period. He also helped me establish my confidence and even the courage to conduct research on such a difficult topic. Secondly, I would like to thank my daily supervisor, Dr. Peng Sun, for his valuable questions and helping pieces of advice.

Many had changed in my life during my thesis period, I would like to thank my parents for their support. And I would also like to thank my dearest friend Boyi for her company and the inspirations she gave rise to. Lastly, I would like to mention Cherry Coffee in Yantai where I spent a lot of time working on this project and produced some good results. The friendly atmosphere and good coffee drinks made my life easier and happier during my summertime in China.

Hanshu Yu
Delft, February 2022

Abstract

We are living in a connected world and failures can occur anywhere at any time probabilistically. In this thesis, we consider networked systems whose links are perfectly reliable and nodes are subject to failure. The probability of a network subjecting to failure to remain connected is named the node reliability of a graph. The node reliability naturally gives rise to a polynomial in the node operational probability p. We call this polynomial node reliability polynomial. The research aims to explore the properties of the node reliability polynomial.

Python tools were developed to compute the exact solutions of node reliability polynomial by enumerating all possible connected sets in graphs. Monte-Carlo simulation software in Python was also developed for approximate solutions of graphs that are too large for enumeration. We took advantage of the developed Python tools to investigate the combinatorics aspect of graphs.

The most important result is that we provide a construction method based on the lexicographic product of graphs such that the node reliability polynomials of two graphs, with the same number of nodes and links, can have an arbitrary number of intersection points. In addition, we have discovered that a fully-joint graph's connected sets are composed by the addition of the connected sets of all partitions along with the connected sets of the complete multi-partite graph that corresponds to the full interconnection between partitions. Later, we propose a conjecture that complete bipartite graphs that are κ-optimal in their class are node reliability optimal in their class. Last but not least, by enumeration of all non-isomorphic graphs of the order less than 10, we have discovered the minimum orders of graph pairs that their node reliability polynomials intersect one, two, and three times. The performance of the crude Monte-Carlo simulation in simulating node reliability polynomial is discussed as well.

Contents

1 INTRODUCTION

Uncertainty is the only certainty there is,
and knowing how to live with insecurity
is the only security.

— *John Allen Paulos*

We are currently living in a connected world.

Connected, not just in the sense between human beings, but in a much much broader sense. People all over the world, are connected by social acquaintances, phones, online social apps, etc. Cities and countries are connected by infrastructures, technical systems are connected by cables and electronics, neurons are connected to form brains, amino acid residues are connected by peptide bonds to form protein molecules, etc. Networks are everywhere and can be used to model almost everything.

Another essential fact of our world, failures occur everywhere at any time. Although undesired, failures are essential considerations for a system. It is often required in networked systems that its components should be able to reach each other within the network for the system to function normally. In this thesis, we study the probability that the networks remaining components are still able to reach each other, or in other words are still connected, under the condition that components can fail probabilistically. Conventionally, we refer to this probability of a network to remain connected after failures occur as network reliability. Reliability can act as a key indicator of the resilience of a network, especially for critical networks like communication networks, infrastructure networks, financial networks, and military networks.

1.1. SETS AND GRAPHS

For the convenience of properly introducing graphs and their connectedness-related topics later in the following chapters, some basic definitions in set theory and graph theory must be introduced beforehand. Here the set theory definitions are placed in the context of von Neumann–Bernays–Gödel set theory (**NBG**). The intention is to provide a relatively rigorous, clear, and concise introduction of basic terms as the foundation of this thesis instead of digging into the details of the terms and definitions.

A **set** is a collection of distinct elements, in which no element is the set itself. If every element in the set $\mathbf{X}$ is an element in the set $\mathbf{Y}$, then set $\mathbf{X}$ is a **subset** of set $\mathbf{Y}$, denoted by $\mathbf{X} \subseteq \mathbf{Y}$. A **null set** is a set with no element, denoted by $\emptyset$.

The purpose of second half of the definition for set is to avoid the famous *Russell's Paradox.*[1] If set is defined with only the first half stating: *a set is a collection of distinct elements*, then naturally in such context a set A can be an element in itself, for instance $\mathbf{A} = \{x, y, z, 1, 2, 3, \mathbf{A}\}$. Following this rule, it is stated that a set containing itself as an element is called an extraordinary set, where the opposite that a set does not contain itself as an element is called an ordinary set. Now, *Russell's Paradox* reveals that the *set* **S** that contains all ordinary sets is neither an ordinary set nor an extraordinary set. Because, assume **S** itself is an ordinary set, it should then contain itself otherwise it is not the set of all ordinary sets. Thus, following the aforementioned statements **S** should be extraordinary and this contradicts the initial assumption that **S** is an ordinary set. It is obvious that **S** is not an extraordinary set following the similar argument.

To deal with this paradox, **class** is introduced to get rid of the extraordinary sets. A simplified definition of **class** can be *a collection of sets that the sets within a class follows a set-level property.* A class that is not a set is called a *proper class* while a class that is a set is called a *small class.* The elements of a class are supposed to be sets only.

One important axiom about class is needed later on when the class of graphs is introduced. The *axiom of limitation of size* states that a class **C** is a proper class if and only if all members of the class of all sets **V** (the von Neumann universe) can be mapped one-to-one into class **C**. To put it in a easily comprehensive way, a 'set' that is too large even the class of all sets **V** can be mapped into that 'set' should be regarded as a *proper class* instead of a 'set'. [2]

After sets and classes are defined, it is time to take a look at graphs and related terms and definitions.

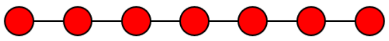

Figure 1.1: Graph example: path graph of order seven $G(7,6)$

A **graph** is a structure that is composed by a set of vertices(nodes) **V** interconnected by a set of edges(links) **E**.[3] Let $N = |\mathbf{V}(G)|$ denote the number of nodes and $L = |\mathbf{E}(G)|$ denote the number of links in graph G. This way a graph G with N nodes and L links can be denoted as $G(N, L)$. Figure 1.1 gives an example graph $G(7,6)$ with 7 nodes and 6 links which is also called an order seven path graph. A graph $H(N', L')$ is called a **subgraph** of $G(N, L)$ if $\mathbf{V}(H) \subseteq \mathbf{V}(G)$ and $\mathbf{E}(H) \subseteq \mathbf{E}(G)$.

A **path** of length $k-1$ from node A to node B in graph G is the node list $P_{A\to B} = n_A \to n_2 \to \ldots \to n_{k-1} \to n_B$, in which $n_i \subseteq \mathbf{V}(G)$, and for any two different nodes n_i and n_j in $P_{A\to B}$, $n_i \neq n_j$. Then, a graph G is called **connected** if there exist a path between any pair of nodes in the graph. Following the connectedness of a graph, the minimum number of removed links to disconnected a graph is called **link connectivity**, denoted as $\lambda(G)$. Similarly, the minimum number of removed nodes to disconnect a graph is called **node connectivity**, denoted as $\kappa(G)$.

In this thesis, it is desired to focus on **connected, unweighted, simple, undirected** graphs. This implies that the links are unweighted, no more than one link can exist between any pair of nodes, no self-loops, and the links have no direction.

Another important concept is called graph isomorphism. It is stated that two graphs G and H are **isomorphic** if there exist a one-to-one mapping between $\mathbf{V}(G)$ and $\mathbf{V}(H)$ that the adjacent property in $\mathbf{V}(G)$ preserves for any node pair after mapped to $\mathbf{V}(H)$ and vice versa. Figure 1.2 gives a simple example of graph isomorphism. In which the nodes can be mapped as follows: $\{a \leftrightarrow p, b \leftrightarrow q, c \leftrightarrow y, d \leftrightarrow x\}$.

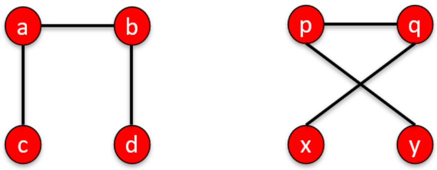

Figure 1.2: An example of isomorphism between two graphs

Finally, one last fundamental concept to complete this section, is the *class* of connected graphs with the same number of nodes and links. We denote this as $\Omega(N, L)$. One might wondering, why this is a class instead of a set. This is because that even for a graph with a certain determined topology, the node set is not specified and can be any. So, we can easily imagine, for the example of graphs with only a single node, the graph with node 1 can be isomorphic to the graph with node $2, 3, 4, 5...$ and infinitely many. According to the *axiom of limitation of size*, such graphs sets (one single graph within each set) should belong to the class $\Omega(1, 0)$. The same argumentation process stands for any class of connected graphs $\Omega(N, L)$ with the same number of nodes N and links L.

1.2. NETWORK RELIABILITY

After the fundamental concepts about graphs and sets are depicted, it is time to dive into the probabilistic graph models. It is assumed that the components, i.e. nodes and edges, should work(operate) or fail at a certain probability. In this thesis, it is assumed that the status of the components are independent of each other and have uniform operating/failing probabilities.

The most commonly researched model of graph reliability is when nodes are perfect in a graph and links would fail independently at a probability $1 - p$. Then the probability of the graph remains connected after link failures is called all-terminal reliability, denoted as $Rel(G, p)$. Here p denotes the link operating probability. When a node has no link attached after the link failure process, the node is removed in the graph. A connected remaining node set with k nodes of graph G after node or link failures is called a **connected set**. We denote the set of all k-ordered connected sets of graph G as

$C_k(G)$. The definition of all-terminal reliability naturally leads to a polynomial.

$$Rel(G,p) = \sum_{k=1}^{|V(G)|} \sum_{S \in C_k(G)} p^{|S|}(1-p)^{|V(G)|-|S|} \tag{1.1}$$

According to [4] the all-terminal reliability is a coherent set system, which indicates that all subsets of a connected set should still remain connected. [5] pointed out that the all-terminal reliability polynomial is monotonously increasing in $(0,1)$. In Figure 1.3, the all-terminal reliability of the order seven path graph is shown as an example. The highest order of the all-terminal reliability of P_7 is 6.

From the computational aspect, it was proven in [6] that computing all-terminal reliability for an arbitrary graph is a NP-hard problem. Yet, many reduction techniques [7, 8, 9, 10] could be applied to reduce the computation time. For series-parallel graphs, the (all-terminal) reliability-preserving series-parallel reductions enables the all-terminal reliability to be computed in linear time [11, 12].

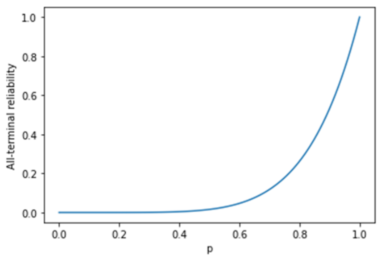

Figure 1.3: All-terminal reliability of order 7 path graph $Rel(P_7,p)$

In this thesis, the focus is on the 'opposite' of all-terminal reliability. It is assumed that links are perfect and nodes are subject to failure with probability $1-p$. The probability that the graph remains connected after node failure is called **node reliability**, denoted as $nRel(G,p)$. Sometimes the node failure reliability model is also called *residual node connectedness reliability*. The node reliability model is first introduced in [13]. Let p denote the node operating probability. The definition of node reliability automatically also leads to a polynomial in p. Compared to the all-terminal reliability polynomial, we alter the equation of node reliability polynomial for it to be easier to use.

$$nRel(G,p) = \sum_{k=1}^{N} c_k(G) p^k (1-p)^{N-k} \tag{1.2}$$

where, $c_k(G)$ represents the total number of connected set of order k: $|C_k(G)|$, we call $c_k(G)$ the ***connected-set coefficient*** *of order k*. When used in the node reliability polynomial for a given graph, c_k is used and '(G)' can be left out for convenience. N is the number of nodes in graph G, while p is the node operating probability.

It is possible to determine some of the connected-set coefficients from graph topological metrics. According to [14], for a connected graph G,

- $c_1(G) = |V(G)|$

- $c_2(G) = |E(G)|$
- $c_3(G) = (\sum_{v \in V(G)} (\binom{d_v}{2})) - 2\Delta$
- $c_{N-1} = N - N_C$
- $c_N = 1$

in which d_v denotes the degree of node v, Δ is the number of triangles in G, N_C is the number of cut nodes of G.

Compared to the all-terminal reliability model, although in a similar model setting, node reliability did not receive as much research attention as all-terminal reliability [15]. Like all-terminal reliability, it is proved in [16] that computing node reliability for an arbitrary graph is also a NP-hard problem. Unfortunately, node reliability does not lead to a coherent system. For example, if all the nodes of degree 2 in a path graph forms a connected set, then it is obvious that not all subsets of this connected sets are connected. This also leads to the non-monotone property of the node reliability polynomial. Again, the node reliability of order-7 path graph in Figure 1.1 is computed. It can be clearly observed in Figure 1.4 that there exist an interval of decrease. According to [4], for a graph G(N,L) that satisfies $L \leq 0.0851N^2$, then the node reliability polynomial $nRel(G,p)$ of this graph G has an interval of decrease in $(0,1)$. Brown[17] provided a construction method and proved that there can be arbitrary number of inflection points as well as decreasing intervals for node reliability polynomials.

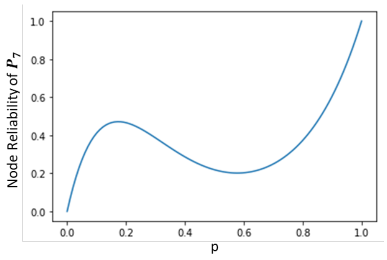

Figure 1.4: Node reliability of order 7 path graph $nRel(P_7,p)$

The (all-terminal) reliability network reduction and factoring algorithms in [7, 8, 9, 10, 11, 12] are not node reliability preserving due to the non-coherency of the node reliability model. To the best of the author's knowledge, there is not much results on reducing the complexity of node reliability. One result worth mentioning is that some restricted graph classes can be computed efficiently in polynomial time such as trees, cographs and permutation graphs [18].

1.3. MAIN CONTRIBUTIONS

The main contributions of this thesis are:

1. Discovered the property of node reliability that when m graphs $\{H_i\}$, $i = 1,2,...,m$, are fully connected together to form a large graph G, the connected-set coefficients

of G equals to the sum of the connected-set coefficients of those m graphs plus the connected-set coefficients of a complete m-partite graph that represents the interconnection.

2. Provides exact solutions for several graph families' node reliability polynomial.

3. A construction method is provided such that it is possible to construct two graphs, with different topology but having the same number of nodes and links, such that their node reliability polynomials can have arbitrary number of intersection points.

1.4. THESIS OUTLINE

This thesis is organized in the following manner:

- Chapter 2 presents the aforementioned first main contribution that fully joint graphs' connected sets are composed by the addition of all partitions with the full interconnection. Then the concept and related works of optimal graphs are introduced, we propose a conjecture that complete bipartite graph that are κ-optimal in their class are node reliability optimal in their class. Later in this chapter we also provide an upper bound for the node reliability polynomial and some exact solutions for some graph families.

- Chapter 3 describes the construction method that two graphs with the same number of nodes and links whose node reliability polynomials can have arbitrary number of intersection points.

- Chapter 4 presents some interesting statistics of brute-forcing non-isomorphic graphs over the graph classes $G(N, L)$ and discusses how crude Monte-Carlo simulation performs in the context of node reliability.

- Chapter 5 summarizes the main achievements of our work and draw the conclusions. Future scope of research are also discussed here.

2

Properties of Node Reliability Polynomials

In this chapter, main topics related to the general property of node reliability polynomials during this thesis research are discussed. Firstly, we briefly discuss the few related works on the node reliability problem. Then we dive into the most important discovery of this chapter. We prove that for fully joint graphs, the connected-set coefficients are computed by summing the connected-set coefficients of m individual partitions with the connected set coefficients of a complete m-partite graph which represents the interconnection between these k individual partitions. Thirdly, we propose a conjecture on the topic of optimal graphs. Lastly, we end this chapter by giving some exact solutions for several graph families.

2.1. Related works with respect to node reliability polynomial

Despite the fact that not much research attention was paid to the node reliability polynomial, there are a few related works and results that are worth mentioning which the reader might find interesting and could be inspired by these results.

2.1.1. Monotonicity and Concavity

Due to the non-coherent nature of the node reliability problem, it is natural that the node reliability polynomials of many graphs are not monotonic, as we have shown in Figure 1.4 as an example. It was proved in [4] that for a graph $G(N, L)$, if G is sufficiently sparse such that $L < 0.0851N^2$, then $nRel(G, p)$ has an interval of decrease in (0,1). However, G is required to be very very dense to ensure that its node reliability polynomial is monotonously increasing in (0,1). Even complete graphs with one pendant node $K_n + P_1$, whose exact formula will be given in section 2.6.2 Eq.(2.17), has an interval of decrease in (0,1). It is known for sure that the node reliability polynomials of complete graphs and complete bipartite graphs are monotonously increasing.

In addition, the node reliability polynomial $nRel(G,p)$ of a graph G is concave for p sufficiently close to 0 if $|V(G)| \geq 2$. When p is close to 1, the node reliability may be convex or concave. The node reliability of any tree is convex when p is close to 1. Thus, one natural extension is that any tree's node reliability polynomial possesses at least one inflection point in (0,1). When a graph $G(N,L)$ is sufficiently sparse that $L < 0.0851N^2$, when $\kappa(G) \geq 2$, $nRel(G,p)$ has at least two distinct points of inflection in (0,1). [4, 19]

It is proven in [17] that the maximum number of decreasing intervals in [0,1] is unbounded for connected graphs, A connected graph with an arbitrary number of inflection points in [0,1] can be constructed with the construction method provided in [17].

2.1.2. Optimal graphs

The search for optimal graphs has been an important topic for link failure system. For node failure situation, the topic also received some attention. Currently, whether optimal graphs exist in $\Omega(N,L)$ for any N and L is far from being known. Here we present some of the known results in literature.

For a graph to be the most reliable when the node operation probability is close to 1, the graph needs to have the largest node connectivity: κ-optimal. When the node operation probability is close to 0, the graph needs to be c_3-optimal to be the most reliable graph. Where c_3-optimal indicates that the graphs should induce the most connected subgraphs of order 3. [20]

Most optimal graphs are found to be complete bipartite graphs. According to Stivaros [21] and Mol [19], star graphs K_{1,n_2} are optimal graphs in their class.Boesch[22] mentioned that Bermond discovered that the almost regular complete bipartite graphs $K_{n,n+1}$ are optimal in their class. Goldschmidt et al.[20] proved the result discovered by Bermond. The almost-almost regular complete bipartite graphs $K_{n,n+2}$ are also proved to be optimal graphs in their class in [20]. When $K_{n,n+b}$ and $b > 2$, the complete bipartite graphs are not optimal graphs in their class. And for sparse classes $\Omega(N,L)$ where $L < N^2/4, L \neq N-1$, optimal graphs do not exist.

Liu et al.[23] proved that complete tri-partite graphs $K_{n,n+1,n+2}$ are optimal in their class. Later, Yu et al.[24] proved that the complete multi-partite graphs $K_{n,n+1,\ldots,n+1,n+i}$ are optimal graphs in their class when $i = 2$. When $i > 2$, the multi-partite graphs $K_{n,n+1,\ldots,n+1,n+i}$ are not uniformly optimal.

2.1.3. A coherent model with respect to node failures

Boesch et al. [25] proposed an alternative definition to node reliability with the motivation to make a coherent system when nodes can fail and links are perfect. Boesch et al. defines a 'k-node operating-component reliability' as the situation that the subgraph of surviving nodes contains at least one connected component having at least k nodes. The 'k-node operating-component reliability' is denoted as $R^k_{oc}(G)$.

In this system, when Boesch proves that when a graph G is connected, and the complement graph G^c do not contain triangles, then this graph G is R^2_{oc} optimal. While the node failure probability $1-p$ is large, the star graph optimizes $R^k_{oc}(G)$. For $n > 4$, the complete bipartite graph (star graph) $K_{1,n-1}$ is $R^{n-1}_{oc}(G)$ and $R^{n-2}_{oc}(G)$ optimal. Lastly, the path graph P_n is uniquely $R^2_{oc}(G)$ optimal.

2.2. GRAPH COMBINATIONS: ADDITIVE CONNECTED SETS OF MULTIPLE FULLY JOINT GRAPH PARTITIONS

In this section, we consider two graph operations: disjoint union and (full) join and their impacts on node reliability. Firstly, we observe the impact through giving examples. Then we propose theorems from the observations the example and give mathematical proofs.

Now we take 2 graphs, H_1 an order 5 path graph and H_2 an order 4 star graph as example. G_1 is the disjoint union of H_1 and H_2, shown in Figure 2.1. G_2 is the fully joint graph of H_1 and H_2, shown in Figure 2.2. The interconnection between P_5 and S_4 can be described by a complete bipartite graph $K_{5,4}$. The connected-set coefficients of P_5, S_4, $K_{5,4}$, G_1, and G_2 are computed and shown in Table 2.1.

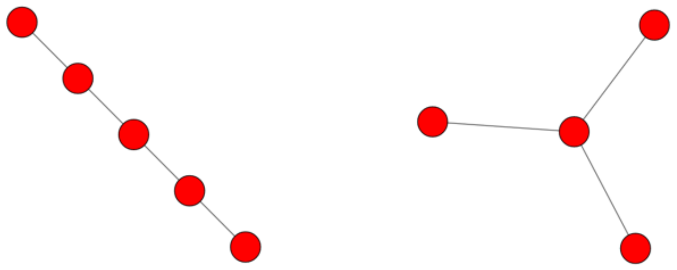

Figure 2.1: $G_1 = P_5 \cup S_4$

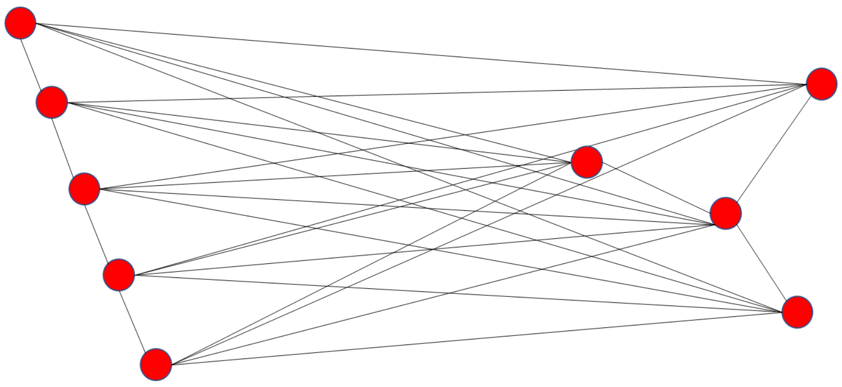

Figure 2.2: $G_2 = P_5 \otimes S_4$

We can observe from Table 2.1 that:

- $c_k(P_5) + c_k(S_4) = c_k(G_1) = c_k(P_5 \cup S_4)$, for all $k = 1,2,...,9$
- $c_k(G_1) + c_k(K_{5,4}) = c_k(G_2)$, for $k = 2,3,...,9$

k	1	2	3	4	5	6	7	8	9
$H_1 = P_5$	5	4	3	2	1	0	0	0	0
$H_2 = S_4$	4	3	3	1	0	0	0	0	0
$K_{5,4}$	9	20	70	120	125	84	36	9	1
$G_1 = P_5 \cup S_4$	9	7	6	3	1	0	0	0	0
$G_2 = P_5 \otimes S_4$	9	27	76	123	126	84	36	9	1

Table 2.1: Connected-set coefficients of P_5, S_4, $K_{5,4}$, G_1, and G_2

Two additive patterns can be discovered. One is the connected-set coefficients of the disjoint union equals to the sum of its disjoint subgraphs. The other is the connected-set coefficients of the fully joint graphs equals to the sum of the connected-set coefficients of the original graphs with their interconnection, which is a complete 2-partite(bipartite) graph $K_{|V(H_1),V(H_2)|}$.

Fortunately, when disjoint uniting or fully joining arbitrary number of graphs, the additive pattern for the connected-set coefficients should still hold.

Theorem 2.1. Given a disconnected graph G constructed by the disjoint union of m simple graphs H_i, $i = 1,2,...m$. $V(H_1) \cap V(H_2) \cap ... \cap V(H_m) = \emptyset$. The connected-set coefficients $c_k(G)$:

$$c_k(G) = \sum_{i=1}^{m} c_k(H_i) \tag{2.1}$$

for all $k = 1,2,...,|V(G)|$. For any graph H, it is defined that $c_k(H) = 0$ if $k > |V(H)|$.

Proof. It is trivial due to the fact that $V(H_1) \cap V(H_2) \cap ... \cap V(H_m) = \emptyset$,

$$C_k(H_1) \cap C_k(H_2) \cap ... \cap C_k(H_m) = \emptyset$$

And the disjoint union of graphs preserves the original connectedness in the separate m graphs and does not introduce new connected sets because no new links are created. $C_k(H_i) \subseteq C_k(G)$ for all i,

$$C_k(G) \setminus \{C_k(H_1) \cup C_k(H_2) \cup ... \cup C_k(H_m)\} = \emptyset$$

And the theorem follows. □

Theorem 2.2. Given a graph G constructed by fully joining m simple graphs H_i, $i = 1,2,...m$. $V(H_1) \cap V(H_2) \cap ... \cap V(H_m) = \emptyset$. The connected-set coefficients $c_k(G)$:

$$c_k(G) = c_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) + \sum_{i=1}^{k} c_k(H_i) \tag{2.2}$$

for $k = 2,...,|V(G)|$. $K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}$ denotes the complete m-partite graph with each partition size corresponds to the order of $H_1, H_2, ..., H_m$. For any graph H, it is defined that $c_k(H) = 0$ if $k > |V(H)|$. While $k = 1$,

$$c_1(G) = \sum_{i=1}^{m} c_1(H_i)$$

Proof. To prove $c_k(G) = c_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) + \sum_{i=1}^{k} c_k(H_i)$, we can examine the inequalities:

1.
$$c_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) + \sum_{i=1}^{k} c_k(H_i) \leq c_k(G) \tag{2.3}$$

2.
$$c_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) + \sum_{i=1}^{k} c_k(H_i) \geq c_k(G) \tag{2.4}$$

The first inequality implies that there should exist a set X consisting of the connected-set of G that X do not intersect with $\{C_k(H_i), i = 1,2,...,m\} \cup C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|})$ for all m partitions in graph G when $k \geq 2$. For every connected-set in X, there are two possibilities:

1. Nodes in any connected-set $S \in X$ are from one single partition $V(H_i)$
2. Nodes in any connected-set $S \in X$ are from multiple partitions $V(H_i) \cup V(H_j) \cup ...$

In the first case, $S \in C(H_i)$, where $S \subseteq V(H_i)$ and $H_i \in \{H_j | j = 1,2,...,m\}$. This conflicts with the condition that $X \cap \{C_k(H_i), i = 1,2,...,m\} = \emptyset$. In the second case, $S \in C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|})$, where $S \subseteq V(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|})$. This conflicts with the condition that $X \cap C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) = \emptyset$.

Since we discover no additional connected-set in the only two possibilities, $X = \emptyset$. But so far this is insufficient to say the inequality does not hold. The final step also relates to the second inequality.

Examining the second inequality, this indicates that,

$$C_k(H_1) \cap C_k(H_2) \cap ... \cap C_k(H_m) \cap C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) \neq \emptyset$$

Because it is given that $C_k(H_1) \cap C_k(H_2) \cap ... \cap C_k(H_m) = \emptyset$, and the connected sets in $C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|})$ when $k \geq 2$ always contain nodes from different partitions, thus

$$C_k(H_1) \cap C_k(H_2) \cap ... \cap C_k(H_m) \cap C_k(K_{|V(H_1)|,|V(H_2)|,...,|V(H_m)|}) = \emptyset \tag{2.5}$$

With Eq.(2.5), neither of the inequalities 2.3 and 2.4 hold. So we must take the equality and this completes the proof.

□

Theorem 2.1 can be used to reduce the computational efforts of disconnected graphs. By computing the connected sets of the connected components, the size of enumeration can be reduced.

Theorem 2.2 can also be applied to reduce the computational efforts of some connected graphs. If a graph can be divided into k components that all k components are fully connected with each other, then the problem is reduced to only computing the connected-sets of these k components and the connected sets of a complete k-partite graph.

However, the graph partitioning process that seeks for full interconnection between partitions is related to the maximum-cut problem. Unfortunately, it is one of the earliest problems that were shown to be NP-hard [26, 27]. To the author's best knowledge, there does not exist an efficient algorithm that reduces the computational complexity of the maximum-cut problem in the general case.

2.3. Conjecture: κ-optimal complete bipartite graphs are optimal in their class

According to [20, 23], an optimal graph in its class requires this graph to be c_3-optimal and κ-optimal. Stivaros [21] states that complete bipartite graphs are c_3-optimal in their class. It is also possible to observe from the formula of c_3:

$$c_3(G) = (\sum_{v \in V(G)} \binom{d_v}{2}) - 2\Delta$$

Where Δ is the number of triangles in a graph. The complete bipartite graphs does not contain any triangle. Here we propose a conjecture:

Conjecture 2.3.1. *κ-optimal complete bipartite graphs are optimal in their class*

Lemma 2.3. *$\Omega(N, L)$ with determined node number $N = n_1 + n_2$ and link number $L = n_1 n_2$, there exist only one unique pair of cardinalities (n_1, n_2) for complete bipartite graph constructions.*

Observation 2.4. *Not all complete bipartite graphs are κ-optimal.*

To illustrate Observation 2.4 and discover under what condition the complete bipartite graphs are κ-optimal, we do the following analysis.

For all graphs, the node connectivity is bounded by the minimum degree.

$$\kappa \le d_{min} \tag{2.6}$$

For complete bipartite graphs K_{n_1,n_2}, assume $n_1 \le n_2$, the node connectivity $\kappa(K_{n_1,n_2}) = n_1$. If we desire the complete bipartite graph to be κ-optimal in its class $\Omega(n_1 + n_2, n_1 n_2)$, one trivial way to guarantee this optimality is:

$$\kappa(G) \le d_{min}^{(G)} \le \kappa(K_{n_1,n_2}) = \min\{n_1, n_2\} = n_1, \quad \forall G \in \Omega(n_1 + n_2, n_1 n_2) \tag{2.7}$$

Considering the degree sequence of complete bipartite graphs,

$$\sum_{v \in V(K_{n_1,n_2})} d_v = 2n_1 n_2$$

The degree sum should not be able to allow K_{n_1,n_2} to be rewired in a way that the minimum degree of the rewired graph is larger than n_1. Because we have assumed $n_1 \le n_2$, the bottom-line requirement for the degree sum is that the sum should not be able to allow the existence of a regular graph with $n_1 + n_2$ nodes of uniform degree distribution $n_1 + 1$, as described in .

$$2n_1 n_2 \le (n_1 + 1)(n_1 + n_2) \tag{2.8}$$

Simplifying inequality Eq.(2.8), we obtain the following:

Lemma 2.4. *Given a complete bipartite graph K_{n_1,n_2} with $n_1 \le n_2$, if n_1 and n_2 satisfy the condition:*

$$n_2 \le \frac{n_1^2 + n_1}{n_1 - 1} = n_1 + \frac{2n_1}{n_1 - 1}, \tag{2.9}$$

then K_{n_1,n_2} is κ-optimal in its class $\Omega(n_1 + n_2, n_1 n_2)$.

Because the cardinalities n_1 and n_2 of complete bipartite graphs must be positive integers. The condition in Eq.(2.9) is asymptotically equivalent to:

$$n_2 \le n_1 + 2, \text{ when } n_1 \ge 3$$

When $n_1 = 1$, all positive n_2 satisfy the condition in inequality Eq.(2.9) because $\left.\frac{2n_1}{n_1-1}\right|_{n_1=1} \to \infty$.
When $n_1 = 2$, $n_2 = n_1 + 3 = 5$ also satisfies inequality Eq.(2.9).

Concerning the known results we mentioned in section 2.1.2, the remaining parts to be proved for condition Eq.(2.9) are:

1. $K_{2,5}$ is optimal in $\Omega(7,10)$
2. regular complete bipartite graphs $K_{n,n}$ are optimal in their class $\Omega(2n, n^2)$

Computer programs are utilized to generate all non-isomorphic connected graphs in $\Omega(7,10)$ for enumeration of the node reliability polynomials. The enumeration result verifies that $K_{2,3}$ is optimal in $\Omega(7,10)$.

Besides, the non-isomorphic connected graph in $\Omega(2n, n^2)$ where $n = 1,2,3,4,5$ are enumerated by the same computer programs. The enumeration result matches the unproved part 2 above. Unfortunately, the number of non-isomorphic connected graphs grows exponentially and it is never possible to enumerate all possible graphs. A proof is not yet know by the author of the thesis. Now we propose the following conjecture:

Conjecture 2.4.1. *Regular complete bipartite graphs $K_{n,n}$ are optimal in their class $\Omega(2n, n^2)$*

Thus conjecture 2.3.1 can be established by proving conjecture 2.4.1.

2.5. AN UPPER BOUND FOR THE NODE RELIABILITY POLYNOMIAL

For an arbitrary connected graph $G(N,L)$, if we enumerate through all possible node combinations, we can obtain a full combination set of order k denoted as $A_k(G)$ where $k = 0,1,\ldots,N$. The connected sets are denoted as $C_k(G)$, and the disconnected sets are denoted as $F_k(G)$.

$$C_k(G) \cup F_k(G) = A_k(G)$$

$$|C_k(G)| + |F_k(G)| = |A_k(G)| \tag{2.10}$$

From the perspective of disconnected sets, we can provide a lower bound for the number of disconnected sets given the degree sequences $\{d_v | v \in V(G)\}$. For an arbitrary node v in graph G, to disconnect this graph, one straightforward decision is to disconnect that node v by failing all v's adjacent nodes. So, if we want to guarantee that node v is in the disconnected set, if d_v is smaller than k, then $k-d_v$ nodes can be selected. Because we are selecting $|V(G)|-k$ nodes from $V(G)$, the worse case is that one set is repeated $|V(G)|-k$ times. A lower bound can be obtained for the disconnected set following the selection process above:

$$|F_k(G)| \geq \frac{1}{N-k} \sum_{v \in V(G)} \binom{N-d_v-1}{k-d_v} \tag{2.11}$$

When $N-k$, we define the result of Eq.(2.11) to be zero instead of ∞.

Before we continue, it is necessary to define a very important alternation to the formula of the binomial coefficient used only in this section.

Definition 2.1. For any integer $K \geq 0$:

$$\binom{K}{0} = 0 \tag{2.12}$$

This alternation in Definition 2.1 in crucial with respect to the actual meaning of graph combinatorics. Selecting zero nodes indeed creates an empty set but this empty set is of no meaning to the counting of sets that cuts the graph (unless the graph itself is disconnected which violates our initial assumption). In Eq.(2.11), $k = d_v$ means that the neighbors of node v already filled a set with k nodes that perhaps would cut the graph, we do not need to choose another node. One may argue that in this case the set of the neighbors of node v should be accounted. But since it is a lower bound of $|F_k(G)|$, removing all neighbors of a node does not guarantee the remaining subgraph is disconnected. One trivial example is the star graph, if all neighbors of the hub node is removed, the remaining subgraph is still considered as connected.

Now we proceed following Eq.(2.11), naturally, the all possible node combinations $|A(G)|$:

$$|A(G)| = \binom{|V(G)|}{k} \tag{2.13}$$

Combining Eq.(2.10), Eq.(2.11) and Eq.(2.13), an upper bound can be obtained.

Theorem 2.5. An upper bound for the k-ordered connected set given the degree sequence $\{d_v | v \in V(G)\}$ of graph G is:

$$|C_k(G)| \leq \binom{|V(G)|}{k} - \frac{1}{N-k} \sum_{v \in V(G)} \binom{N-d_v-1}{k-d_v} \tag{2.14}$$

Our bound corresponds to the result in [20]. The equality holds for Eq.(2.11) and Eq.(2.14) only when there does not exist a set of nodes that does not contain node v and not all the adjacent nodes of v that cuts the graph G for any $v \in V(G)$.

We end this section by providing some examples in Table 2.2 to illustrate the upper bound in Eq.(2.14).

Graph	Path graph P_5		Star graph S_5		Cycle graph C_5		Complete bipartite graph $K_{2,5}$		Complete graph K_5	
Degree Distribution	{1,2,2,2,1}		{4,1,1,1,1}		{2,2,2,2,2}		{2,2,3,3,3}		{4,4,4,4,4}	
Order k	$\lvert C_k \rvert$	Upper Bound	$\lvert C_k \rvert$	Upper Bound	$\lvert C_k \rvert$	Upper Bound	$\lvert C_k \rvert$	Upper Bound	$\lvert C_k \rvert$	Upper Bound
5	1	1	1	1	1	1	1	1	1	1
4	2	5	4	5	5	5	5	5	5	5
3	3	10	6	10	5	10	9	10	10	10
2	4	8	4	6	5	10	6	10	10	10
1	5	5	5	5	5	5	5	5	5	5

Table 2.2: Exact number of k-ordered connected sets versus the upper bound in Eq.(2.14) for some connected graphs with order 5

2.6. EXACT SOLUTIONS

In this section exact solutions are computed for some graph families.

2.6.1. COMPLETE GRAPH

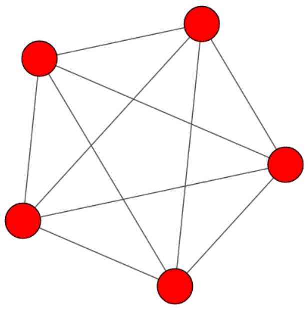

Figure 2.3: Complete graph of order 5 K_5

The node reliability polynomial equation for the complete graph (denoted as K_n) is given as follows:

$$nRel(K_n, p) = 1 - (1-p)^n \tag{2.15}$$

It is obvious for complete graphs where all nodes are connected to each other, that as long as there is one operating node, the graph stays connected. It is only possible to disconnect this graph by failing all nodes, which transforms this graph to a null graph.

An example K_5 is given in Figure 2.3, whose node reliability polynomial is depicted in Eq.(2.16) and Figure 2.4.

$$nRel(K_5, p) = p^5 - 5p^4 + 10p^3 - 10p^2 + 5p \tag{2.16}$$

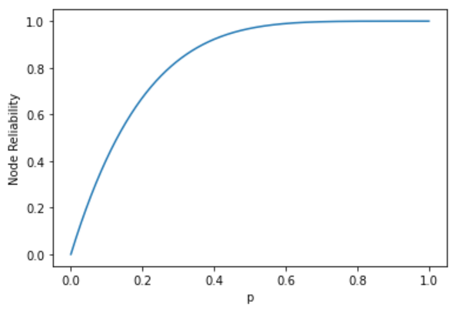

Figure 2.4: A plot of node reliability polynomial of graph K_5

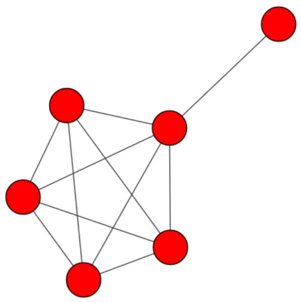

Figure 2.5: Complete graph of order 5 with one pendant node K_5+P_1

2.6.2. Complete graph with one pendant node

The node reliability polynomial equation for complete graph with a pendant node(denoted as K_n+P_1) is given as follows:

$$nRel(K_n+P_1,p) = (1-p)(1-(1-p)^n)+p^2+p(1-p)^n \tag{2.17}$$

When that pendant node fails, we are left with the complete graph K_n. When the pendant node and the adjacent node are both operating, then the graph remains connected no matter the status of other nodes. When the pendant node is operating and the adjacent node is not operating, it is required that none of the other nodes in the complete graph K_n are operating for the entire graph to remain connected.

An example K_5+P_1 is given in Figure 2.5, whose node reliability polynomial is depicted in Eq.(2.18) and Figure 2.6.

$$nRel(K_5+P_1,p) = -2p^6+11p^5-25p^4+30p^3-19p^2+6p \tag{2.18}$$

2.6.3. Star graph

The node reliability polynomial equation for the star graph of order n (denoted as S_n) is given as follows:

$$nRel(S_n,p) = p+(n-1)p(1-p)^{(n-1)} \tag{2.19}$$

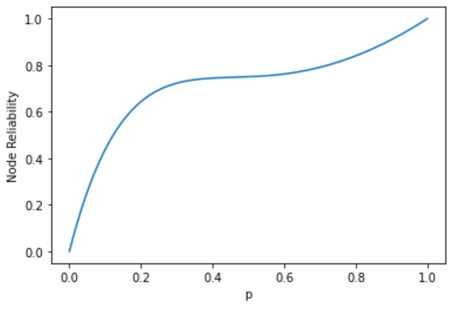

Figure 2.6: A plot of node reliability polynomial of graph $K_5 + P_1$

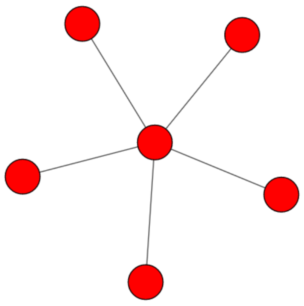

Figure 2.7: Star graph of order 6 S_6

When the hub node in star graph is operating, the graph remains connected regardless of the status of other leaf nodes. When the hub fails, there should be only one leaf node that is operating and other leaf nodes should fail for the graph to remain connected. There are in total $n-1$ leaf nodes.

An example S_6 is given in Figure 2.7, whose node reliability polynomial is depicted in Eq.(2.20) and Figure 2.8.

$$nRel(S_6, p) = -5p^6 + 25p^5 - 50p^4 + 50p^3 - 25p^2 + 6p \tag{2.20}$$

2.6.4. PATH GRAPH

The node reliability polynomial equation for the path graph of order n (denoted as P_n) is given as follows:

$$nRel(P_n, p) = \frac{np(1-p)^{n+1} - (n+1)p^2(1-p)^n + p^{n+2})}{(2p-1)^2} \tag{2.21}$$

An example P_9 is given in Figure 2.9, whose node reliability polynomial is depicted in Eq.(2.22) and Figure 2.10.

$$nRel(P_9, p) = 5p^9 - 40p^8 + 150p^7 - 320p^6 + 430p^5 - 372p^4 + 203p^3 - 64p^2 + 9p \tag{2.22}$$

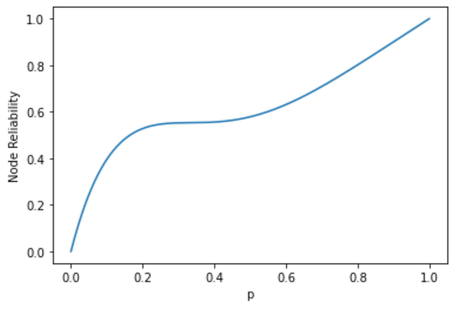

Figure 2.8: A plot of node reliability polynomial of graph S_6

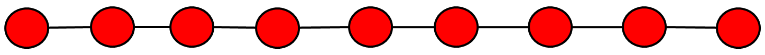

Figure 2.9: Path graph of order 9 P_9

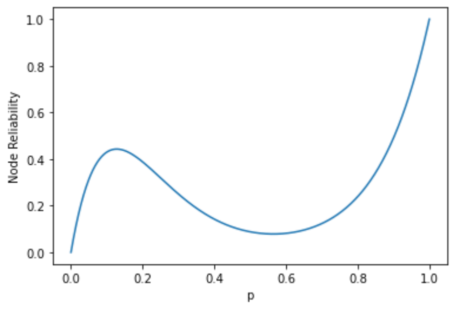

Figure 2.10: A plot of node reliability polynomial of graph P_9

2.6.5. CYCLE GRAPH

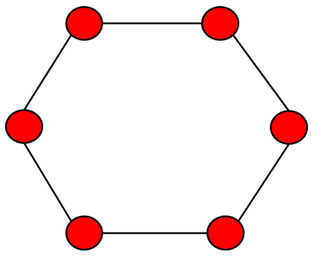

Figure 2.11: Cycle graph of order 6 C_6

The node reliability polynomial equation for the cycle graph of order n (denoted as C_n) is given as follows:

$$nRel(C_n, p) = \frac{np(p^n - (1-p)^n)}{2p-1} - (n-1)p^n \tag{2.23}$$

An example C_6 is given in Figure 2.11, whose node reliability polynomial is depicted in Eq.(2.24) and Figure 2.12.

$$nRel(C_6, p) = -5p^6 + 18p^5 - 36p^4 + 42p^3 - 24p^2 + 6p \tag{2.24}$$

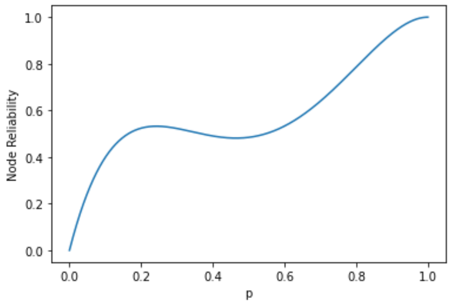

Figure 2.12: A plot of node reliability polynomial of graph C_6

2.6.6. CYCLE GRAPH WITH ONE PENDANT NODE

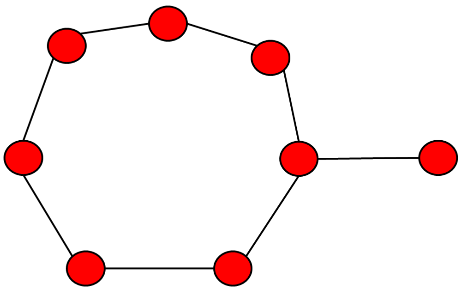

Figure 2.13: Cycle graph of order 7 with one pendant node $C_7 + P1$

The node reliability polynomial equation for the cycle graph of order n with one pendant node (denoted as $C_n + P_1$) is given as follows:

$$\begin{aligned} nRel(C_n + P_1, p) = \\ (1-p)\Big(\frac{np(p^n - (1-p)^n)}{2p-1} - (n-1)p^n\Big) + p(1-p)^n \\ + \frac{p^2}{(2p-1)^2}((2pn - n + p - 1)p^n + (1-p)^{n+1}) - (n-1)p^{n+1} \end{aligned} \tag{2.25}$$

An example $C_7 + P_1$ is given in Figure 2.13, whose node reliability polynomial is depicted in Eq.(2.26) and Figure 2.14.

$$nRel(C_7 + P_1, p) = -4p^8 + 29p^7 - 99p^6 + 181p^5 - 195p^4 + 129p^3 - 48p^2 + 8p \tag{2.26}$$

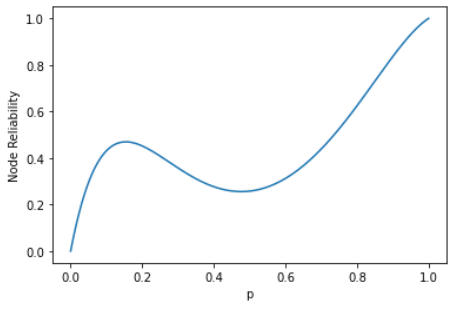

Figure 2.14: A plot of node reliability polynomial of graph $C_7 + P_1$

2.6.7. COMPLETE BIPARTITE GRAPH

The node reliability polynomial equation for complete bipartite graph $K_{n,m}$)is given as follows:

$$nRel(S_n, p) = (1-(1-p)^n)(1-(1-p)^m) + (n+m)p(1-p)^{(n+m-1)} \tag{2.27}$$

The graph remains connected either at least one node in each partition do not fail, or only one node in the entire graph is operating and the others all fail.

2.6.8. DISJOINT UNION OF GRAPHS

Consider two graphs G_1 and G_2 that are combined together without interconnection to form a disconnected new graph $G_1 \cup G_2$.

$$nRel(G_1 \cup G_2, p) = nRel(G_1, p)\,(1-p)^{|V(G_2)|} + nRel(G_2, p)\,(1-p)^{|V(G_1)|} \tag{2.28}$$

The new graph is connected only when one partition's node all fail and the surviving nodes of the other partition induce a connected subgraph.

2.6.9. FULLY JOINT GRAPHS

Consider two graphs G_1 and G_2 that are combined together with full interconnections between $V(G_1)$ and $V(G_2)$ to form a connected new graph $G_1 \otimes G_2$.

$$nRel(G_1 \otimes G_2, p) =$$

$$1-(1-nRel(G_1,p))(1-p)^{|V(G_2)|}-(1-nRel(G_2,p))(1-p)^{|V(G_1)|}+(1-p)^{|V(G_1)|+|V(G_2)|} \tag{2.29}$$

Because we are looking at complete connection between two graphs, the graph is disconnected either all nodes are failed or one partition's node completely fail the the other part's surviving nodes does not induce a connected graph.

3 Intersections of Node Reliability Polynomials

In this section, a construction method is shown such that we can construct 2 graphs with the same numbers of nodes and edges while their node reliability polynomials have an arbitrary number of intersections.

When considering the system of link failures and perfect nodes, Brown[28] shows that the all-terminal reliability polynomials can have an arbitrary number of intersection points. although the two systems have different coherency, the analogy between the definition of all-terminal reliability and node reliability brings the question:

Can node reliability polynomials of 2 graphs, with the same number of nodes and links, intersect an arbitrary number of times?

Brown [17] proved there can be arbitrarily many inflection points in (0,1) for node reliability polynomials. This result made it theoretically possible for node reliability polynomials to intersect an arbitrary number of times. Then the following problem is how to construct such two graphs.

Since node reliability polynomials and graphs are not one-to-one correspondence, it is not hard to find 2 graphs mapped to the same node reliability polynomial coefficients. One example graph pair is given in Figure 3.1. The exact node reliability polynomial of the graph pair is depicted in Eq.(3.1) and Figure 3.2. We can say that these graphs have infinitely many intersection points.

$$nRel(C_4 + P_1) = nRel(C_3 + 2 \times P_1) = 3p^5 - 13p^4 + 21p^3 - 15p^2 + 5p \tag{3.1}$$

We are more interested in the case of finite number of intersections. To generate intersections, one may think of one of the seven requirements for defining the intersection numbers of plane curves. The intersection numbers should add when

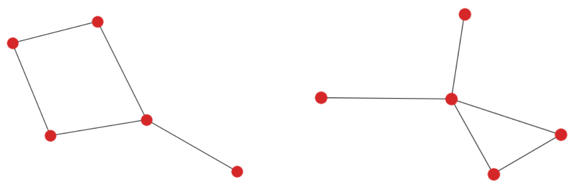

Figure 3.1: Two graphs with the same number of nodes and links that maps to the same node reliability polynomial

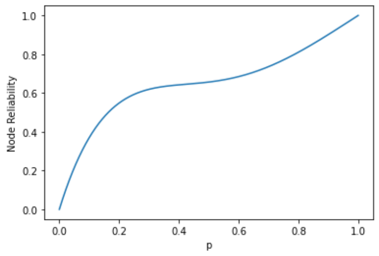

Figure 3.2: A plot of the node reliability polynomial of the example graph pair in Figure 3.1

we take unions of curves[29]. Taking unions of curves from the polynomial aspect refers to the multiplication of polynomials. However, when we place our context in the node reliability polynomial, we find no known graph operation that directly involves multiplying the node reliability polynomials. If we inspect the graph combination solutions given in section 2.6, the disjoint union does not provide a connected graph and the fully joint union process is more 'additive' than 'multiplicative'.

To clearly illustrate how this graph union problem keeps us from creating intersections, we consider 6 connected graphs $G_1(N_1,L_1)$, $G_2(N_2,L_2)$, $G_3(N_3,L_3)$, $H_1(N_1,L_1)$, $H_2(N_1,L_1)$, $H_3(N_1,L_1)$. In each pair of graphs, G_1 and H_1, G_2 and H_2, G_3 and H_3, have the same number of nodes and links in order to guarantee we eventually have 2 graphs with same number of total nodes and links. Now we perform the fully joint process:

$$G = G_1 \otimes G_2 \otimes G_3$$

$$H = H_1 \otimes H_2 \otimes H_3$$

According to Eq.(2.29), one can easily derive:

$$\begin{gathered} nRel(G,p) - nRel(H,p) = \\ (nRel(G_1,p) - nRel(H_1,p))(1-p)^{N_2+N_3} \\ +(nRel(G_2,p) - nRel(H_2,p))(1-p)^{N_1+N_3} \\ +(nRel(G_3,p) - nRel(H_3,p))(1-p)^{N_1+N_2} \end{gathered}$$

It is even difficult to retain the existing intersection points even if the node reliability polynomials in the above mentioned 3 graph pairs intersect at the same position. It is an almost impossible task to generate more intersection points by only fully joining more graphs to existing graphs. The multiplication of the exponent of $(1-p)$ made satisfying $nRel(G,p) - nRel(H,p) = 0$ difficult.

Now, we propose one construction method based on the lexicographic product of graphs.

Definition 3.1. The **lexicographic product** of graph G with graph H is defined to be replacing every node in graph G by graph H, denoted by $G[H]$. If 2 nodes $v_1, v_2 \in V(G)$ are adjacent, then the nodes in the copies of H that replaces v_1 and v_2 have full interconnection.

Concerning the lexicographic product, we expand *Lemma 1* in [17]:

Theorem 3.1. For any connected graphs G and H,

$$nRel(G[H],p) = nRel(G, nRel(H,p)) \tag{3.2}$$

Proof. Consider a surviving non-empty node set $S \subseteq V(G[H]$, then the subgraph induced by S is connected if and only if the graph copy that is replacing each node in $V(G[H])$ is connected. □

Lemma 1 in [17] is a special case of Theorem 3.1 with H being the complete graph K_m.

Although there does not exist a known direct multiplicative relation for graph operations in the context of node reliability, we can easily factor $nRel(G[H],p)$ and create a multiplicative relation from the lexicographic product of graphs. Because when all nodes in a graph fail, we consider the graph fail, so the constant coefficients c_0 in node reliability polynomials are always zero. Here for the convenience of readers we define a function X for any graph G:

$$X(G,p) = \frac{nRel(G,p)}{p}$$

Obviously, by factoring out a p, the function X retains the numbers and types of intersections of node reliability polynomials for determined graph pairs. Then naturally, when we perform the lexicographic product of graph G_0 with graph H_1, we can obtain the node reliability polynomial of G_0 and $G_1 = G_0[H_1]$:

$$nRel(G_0,p) = pX(G_0,p)$$

$$nRel(G_1,p) = nRel(G_0[H_1],p) = nRel(G_0, nRel(H_1,p))$$

$$= nRel(H_1,p)X(G_0, nRel(H_1,p)) = pX(H_1,p)X(G_0, nRel(H_1,p))$$

If we continue this process, let $G_2 = G_1[H_2]$ and we can factor another p from $nRel(H_1,p)$:

$$nRel(G_2,p) = nRel(G_1[H_2],p)$$

$$= nRel(H_2,p)X(H_1, nRel(H2,p))X(G_0, nRel(H_1, nRel(H2,p)))$$

$$= pX(H_2,p)X(H_1, nRel(H2,p))X(G_0, nRel(H_1, nRel(H2,p)))$$

A multiplicative relation of polynomials can be established in this way, representing the union of certain curves. If we can construct two graphs with lexicographic product operations that have the same number of nodes and links, it is hopeful that we can construct more intersection points between graphs. We then take a different graph G'_0 and conduct the lexicographic operation with H'_1, such that $G'_1 = G'_0[H'_1]$. Then we continue to replace every node in G'_1 with H'_2 to obtain $G'_2 = G'_1[H'_2]$. To enlarge the chance that we produce more intersection points, the node reliability polynomials of H_i and H'_i graphs with the same i need to intersect with each other.

Observation 3.1. *Cycle graph C_N and cycle graph with one pendant node $C_{N-1}+P_1$ have one intersection in the interval (0,1).*

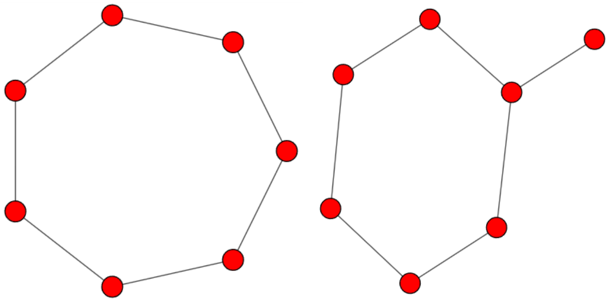
Figure 3.3: C_7 and C_6+P_1

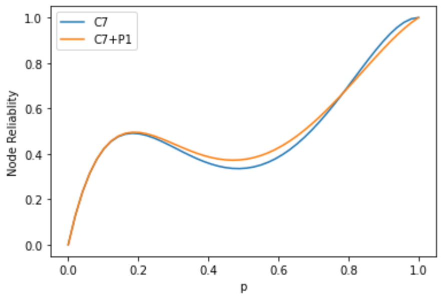

Figure 3.4: The node reliability of C_7 and C_6+P_1

$$nRel(C_7, p) = p^7 - 21p^6 + 63p^5 - 91p^4 + 77p^3 - 35p^2 + 7p \tag{3.3}$$

$$nRel(C_6 + P_1, p) = 4p^7 - 26p^6 + 66p^5 - 93p^4 + 78p^3 - 35p^2 + 7p \tag{3.4}$$

The cycle graph of order 7 and the cycle graph of order 6 plus one pendant node is displayed as an example in Figure 3.3. Their node reliability polynomials are plotted in Figure 3.4, the exact formulae are depicted in Eq.(3.3) and Eq.(3.4). We can spot at approximately $p = 0.7839$ their node reliability polynomials intersect once. For C_N and

N	Position of intersection	N	Position of intersection
5	0.5	13	0.9082
6	0.7071	14	0.9160
7	0.7839	15	0.9225
8	0.8256	16	0.9281
9	0.8528	17	0.9330
10	0.8723	18	0.9372
11	0.8871	19	0.9409
12	0.8987	20	0.9443

Table 3.1: Intersection points between C_N and $C_{N-1}+P_1$

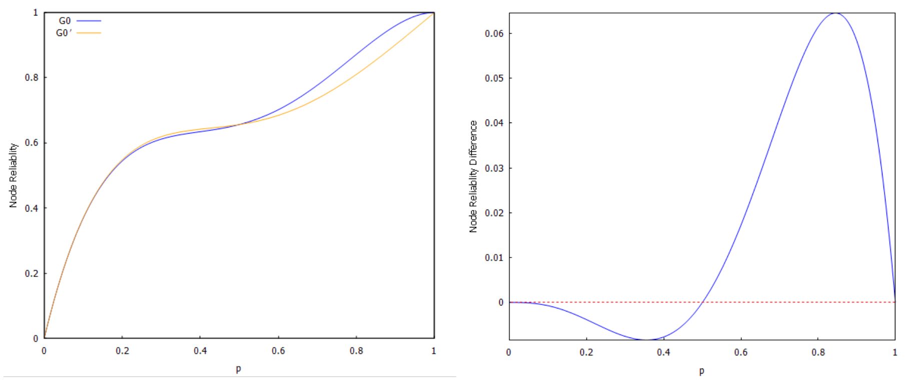

Figure 3.5: Node reliability polynomials of $G_0 = C_5$ and $G'_0 = C_4 + P_1$ intersect once

$C_{N-1}+P_1$ of order $N \in [5,20]$, the positions of intersections are listed in Table 3.1. As the order increases, the intersection points between C_N and $C_{N-1}+P_1$ become closer to 1.

Now we present two examples:

1. we started from 2 intersecting graphs $G_0 = C_5$ and $G'_0 = C_4 + P_1$ and created graph pairs with an odd number of intersections
2. we started from 2 non-intersecting graphs and created graph pairs with an even number of intersections.

Example 3.1. In this first example, we start from 2 intersecting graphs $G_0 = C_5$ and $G'_0 = C_4 + P_1$. Table 3.1 specifies that they have one intersection point at $p = 0.5$ as shown in Figure 3.5. Their exact node reliability polynomials are depicted in Eq.(3.5) and Eq.3.6.

$$nRel(C_5, p) = nRel(G_0, p) = p^5 - 10p^4 + 20p^3 - 15p^2 + 5p \tag{3.5}$$

$$nRel(C_4 + P_1, p) = nRel(G'_0, p) = 3p^5 - 13p^4 + 21p^3 - 15p^2 + 5p \tag{3.6}$$

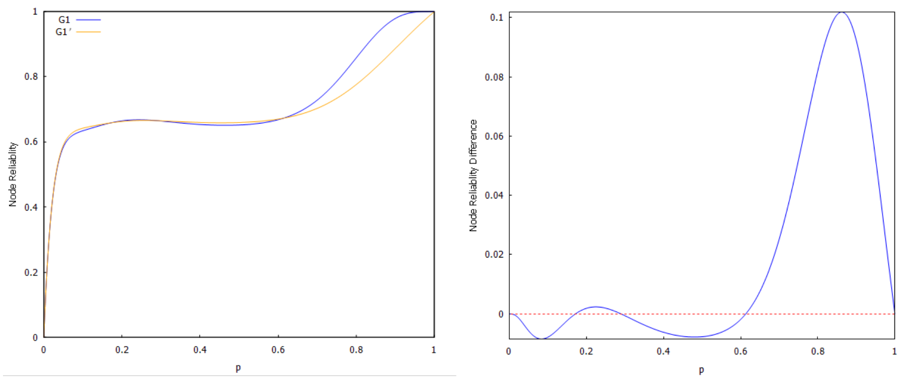

Figure 3.6: Node reliability polynomials of G_1 and G_1' intersect 3 times

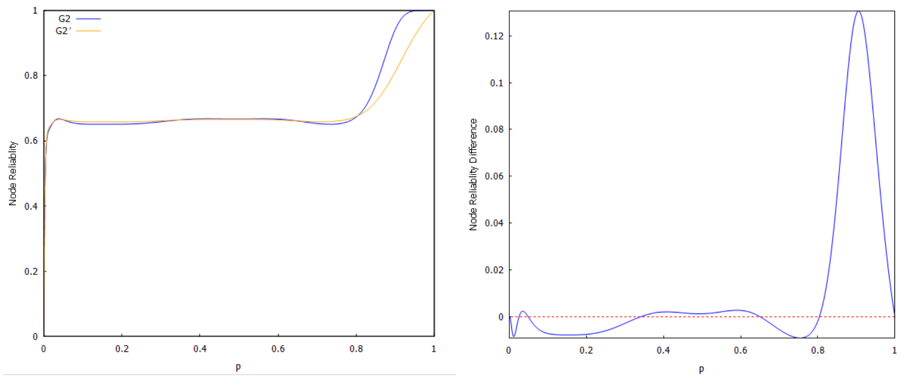

Figure 3.7: Node reliability polynomials of G_2 and G_2' intersect 5 times

We begin the first lexicographic product by setting:

$$H_1 = C_6\,,\; H_1' = C_5 + P_1$$

Thus,

$$G_1 = G_0[H_1]\,,\; G_1' = G_0'[H_1']$$

In Figure 3.6, the node reliability polynomials of $G_1 = G_0[C_6]$ and $G_1' = G_0'[C_5 + P_1]$ are plotted. We can clearly spot there are 3 intersection points at $p = 0.168\,, 0.291\,, 0.613$.

Continuing with the lexicographic product by:

$$G_2 = G_1[C_8]\,,\; G_2' = G_1'[C_7 + P_1]$$

In Figure 3.7, we can see that the node reliability polynomials intersect 5 times at $p = 0.0249\,, 0.0494\,, 0.340\,, 0.650\,, 0.804$.

We repeat the process again with:

$$G_3 = G_2[C_7]\,,\; G_3' = G_2'[C_6 + P_1]$$

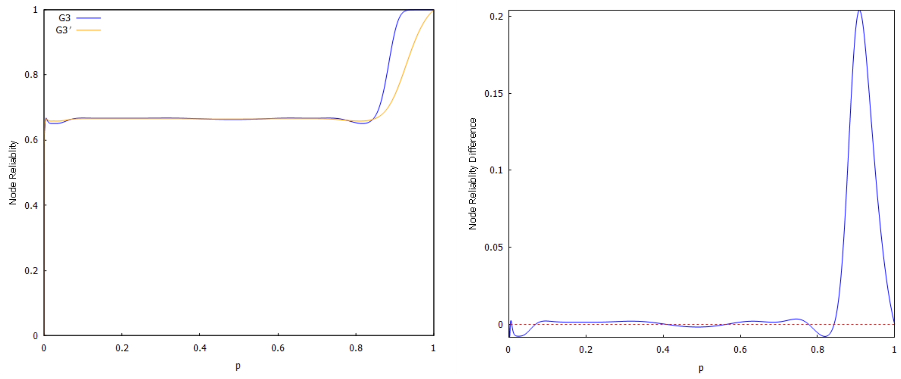

Figure 3.8: Node reliability polynomials of G_3 and G_3' intersect 7 times

Lexicographic operations	$\lvert V(G_i)\rvert$	$\lvert V(H_{i+1})\rvert$	$\lvert V(G_{i+1})\rvert$
1st operation	5	6	30
2st operation	30	8	240
3st operation	240	7	1680

Table 3.2: Graph sizes in Example 3.1

Figure 3.8 shows that now the node reliability polynomials of G_3 and G_3' have 7 intersection points, which are $p = 0.003, 0.007, 0.069, 0.400, 0.566, 0.776, 0.843$.

It is believed that starting from a graph pair whose node reliability polynomials have one intersection point, and repeating such lexicographic product operations with C_N and $C_{N-1} + P_1$ that have an 'appropriate size' of N, we can generate 2 more intersection points in each operation. Thus we can create two graphs, with the same number of nodes and links, depicted in Table 3.2, that their node reliability polynomials have an arbitrary odd number of intersection points.

Example 3.2. In this second example, we start from two 5-node-5-link graphs G_0 and G_0', as shown in Figure 3.9, whose node reliability polynomials do not intersect, see Figure 3.10. The exact node reliability polynomials of G_0 and G_0' are shown in Eq.(3.7) and Eq.(3.8)

$$nRel(G_0) = 3p^5 - 12p^4 + 20p^3 - 15p^2 + 5p \tag{3.7}$$

$$nRel(G_0') = 2p^5 - 10p^4 + 19p^3 - 15p^2 + 5p \tag{3.8}$$

Similarly, as in the first example, we conduct the lexicographic products as follows:

- Lexicographic product 1:

$$H_1 = C_8 \ , \ H_1' = C_7 + P_1$$

$$G_1 = G_0[H_1] = G_0[C_8] \ , \ G_1' = G_0'[H_1'] = G_0'[C_7 + P_1]$$

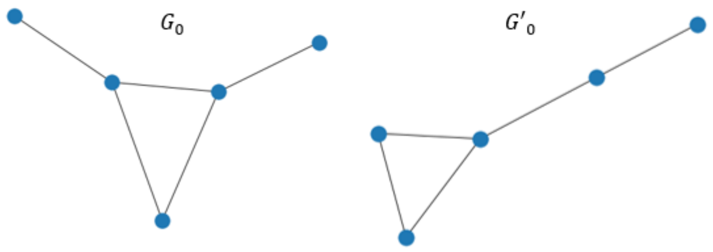

Figure 3.9: G_0 and G'_0, two 5-node-5-link graphs whose node reliability polynomial do not intersect

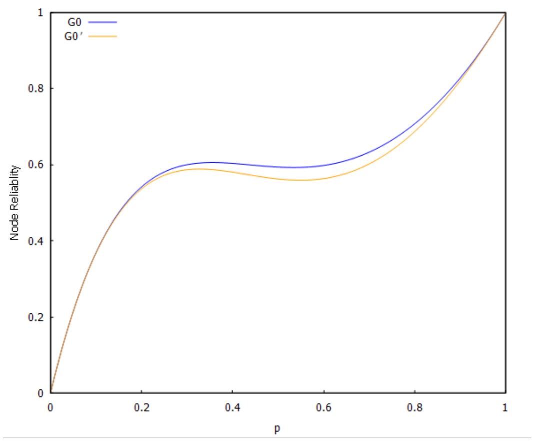

Figure 3.10: Node reliability polynomial of G_0 and G'_0

Figure 3.11 shows that node reliability polynomials of G_1 and G'_1 have 2 intersection points at approximately $p = 0.4050, 0.5919$.

- Lexicographic product 2:

$$H_2 = C_7 \ , \ H'_2 = C_6 + P_1$$

$$G_2 = G_1[H_2] = G_1[C_7] \ , \ G'_2 = G'_1[H'_2] = G'_1[C_6 + P_1]$$

Figure 3.12 shows that node reliability polynomials of G_2 and G'_2 have 4 intersection points at approximately $p = 0.0937, 0.3172, 0.6369, 0.7467$.

- Lexicographic product 3:

$$H_3 = C_7 \ , \ H'_3 = C_6 + P_1$$

$$G_3 = G_2[H_3] = G_2[C_7] \ , \ G'_3 = G'_2[H'_3] = G'_2[C_6 + P_1]$$

Figure 3.13 shows that node reliability polynomials of G_3 and G'_3 have 6 intersection points at approximately:
$p = 0.0153, 0.0124, 0.4525, 0.5192, 0.7683, 0.8213$.

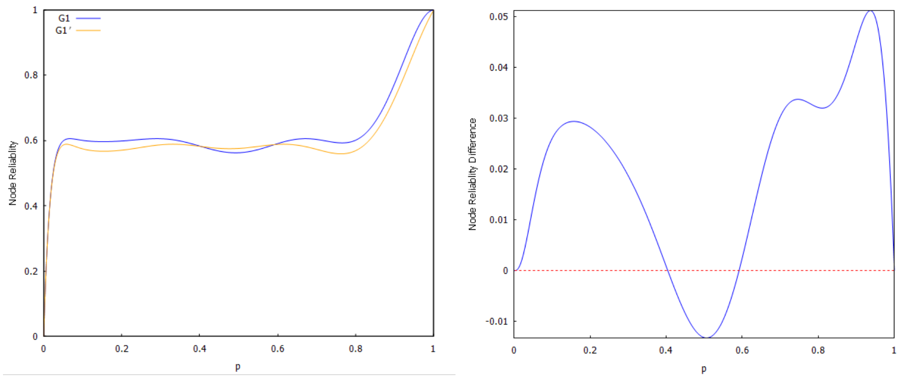

Figure 3.11: Node reliability polynomials of G_1 and G_1' intersect 2 times

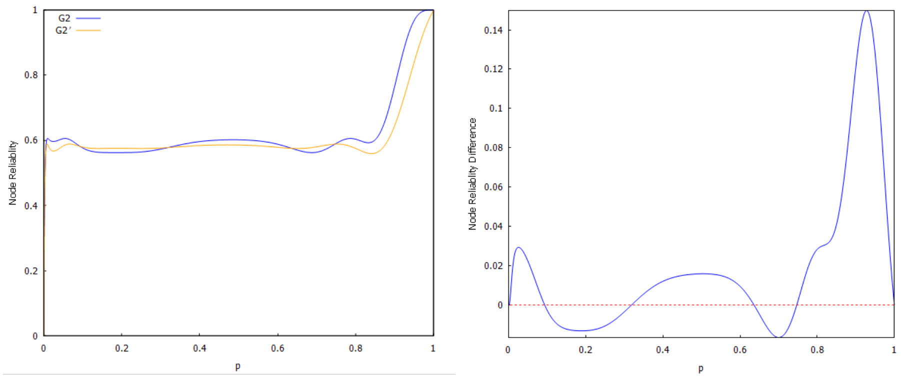

Figure 3.12: Node reliability polynomials of G_2 and G_2' intersect 4 times

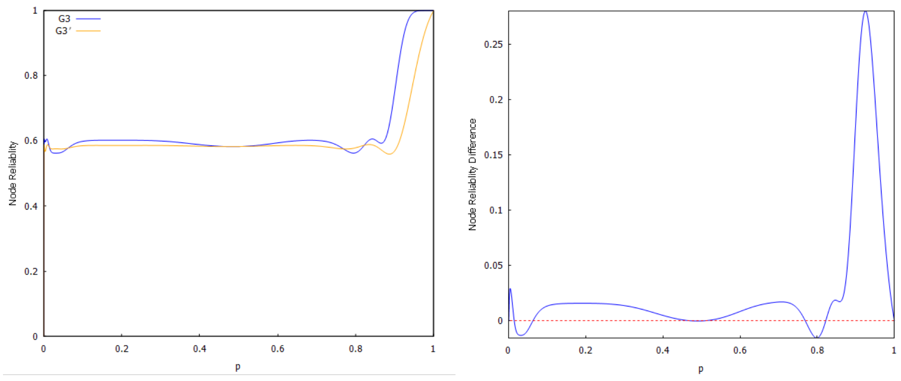

Figure 3.13: Node reliability polynomials of G_3 and G_3' intersect 6 times

Lexicographic operations	$\lvert V(G_i)\rvert$	$\lvert V(H_{i+1})\rvert$	$\lvert V(G_{i+1})\rvert$
1st operation	5	8	40
2st operation	40	7	280
3st operation	280	7	1960

Table 3.3: Graph sizes in Example 3.2

From the results of this example, it is believed that starting from a graph pair whose node reliability polynomials do not intersect, and repeating such lexicographic product operations with C_N and $C_{N-1} + P_1$ that have a 'appropriate size' of N, we can also generate 2 more intersection points in each lexicographic operation. Thus we can create two graphs, with the same number of nodes and links, depicted in Table 3.3, such that their node reliability polynomials have an arbitrary even number of intersection points.

4 Node Reliability via Enumeration and Simulation

In this chapter, using the Python tool developed by the author, we first present some interesting results in brute-force enumeration of small graphs classes. Later in this chapter, we apply crude Monte-Carlo(CMC) simulation to node reliability problems. The performance of CMC is evaluated in the context of simulating node reliability problems.

4.1. Enumeration Results

With tools in nauty & traces [30], we are able to efficiently generate all non-isomorphic graph constructions in a graph class $\Omega(N, L)$. The self-developed Python enumeration tool enumerates all possible subsets of an inputted graph's node set. The connected subgraphs induced by these node subsets are counted with respect to the order, such that we can obtain the connected-set coefficients c_k for all possible orders k. Applying the binomial theorem, the enumeration tool finally gives the node reliability polynomial of a graph. We have enumerated the node reliability polynomials of all non-isomorphic graphs of $N \leq 10$.

In Chapter 3, we have discussed about the intersections of node reliability. In the enumeration process, we have discovered the smallest graph pairs that have 1, 2, and 3 intersections points presented in Table 4.1. We know from the enumeration that when graph size is smaller than 5, there does not exist a graph pair, with the same number of nodes and links, whose node reliability polynomial intersect.

Intersection points	Minimum graph size
1 intersection	5 nodes 5 links
2 intersections	8 nodes 7 links
3 intersections	9 nodes 12 links

Table 4.1: Smallest graph pairs that have 1, 2, and 3 intersection points

Taking $\Omega(7,10)$ and $\Omega(10,12)$ as example, we provide some statistics about graph class in Table 4.2.

	$\Omega(7,10)$	$\Omega(10,12)$
Number of non-isomorphic graphs	132	8548
Number of inflection points	1: 54; 2: 78	1: 6115; 2: 2433
Number of local extrema	0: 30; 2: 102	2: 8548
Number of fixed points	0: 88; 1: 44	0: 2; 1: 8424; 2: 122
Number of intersection points*	0: 6652; 1: 1994	0: 42581; 1: 16919; 2: 200

Number of intersection points: the number of intersection points in $\Omega(10,12)$ is not the full data. The number of all possible graph pairs in $\Omega(7,10)$ is $\binom{132}{2} = 8646$ while in $\Omega(10,12)$ there are $\binom{8548}{2} = 36529878$ possible pairs. Due to the large computational load of $\Omega(10,12)$, we have sampled 200 graphs from all 8548 non-isomorphic graphs and inspected the number of intersection points. The process repeated 3 times with no repeatedly sampled graph in order to obtain more trustworthy statistics. The total number of graph pairs considered is $\binom{132}{2} \times 3 = 59700$.*

Table 4.2: Statistics about graph classes $\Omega(7,10)$ and $\Omega(10,12)$

Over the entire simulation, we have observed that the number of non-isomorphic graphs increases exponentially as the graph order increases. And as the graph order increases, the behavior of graphs becomes richer: more fixed points, more inflection points, more intersection points, more extrema.

4.2. CRUDE MONTE-CARLO SIMULATION IN NODE RELIABILITY

We have implemented a CMC simulator in Python with the *iGraph* package. The simulation is run on a moderate performance laptop with Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz and 12GB RAM. The simulation platform was chosen to be jupyter notebook on a Windows 10 operating system in convenience of the author. To understand the basic performance of the CMC simulation, we have set the graph order to be 20 nodes. Due to the limited computational power and the exponential growth of the enumeration time needed with respect to the number of nodes in a graph, 20 nodes is a size that is relatively large and stable enough to run on the author's computer. Besides, we have simulated different graph families: star graph, path graph, ring(cycle) graph, full(complete) graph, and Erdős–Rényi random graph when the link probability is set to be $\frac{1.05 \dot{l}og20}{20}$, slightly above the critical probability to ensure that the random graph is connected but not too dense. A plot of the node reliability polynomials of the above mentioned graphs of order 20 is presented in Figure 4.1.

We have sampled 19 'p' values in the simulation: 0.1, 0.2, ...,0.8, 0.9, 0.91, 0.92, ..., 0,98, 0.99. Because in real-life networks, usually the components are designed to be very reliable hence we densify the samples in $(0.9,1)$. The rounds of simulation is also a usual concern in Monte-Carlo simulation, hence we have 41 different simulation round samples in the range (1e2,1.25e5). Due to the random nature of many aspects of this CMC simulation process, the experiment is repeated 10 times and here in figures 4.2, 4.5, 4.3, 4.4, 4.8, 4.6, 4.7 we present the averaged results.

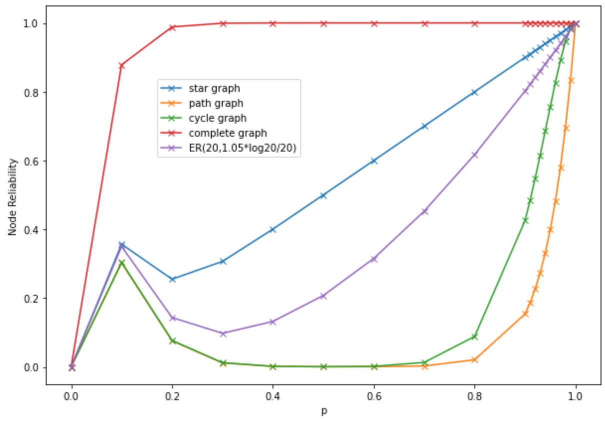

Figure 4.1: Sampled true values of the node reliability polynomials of the simulated graphs

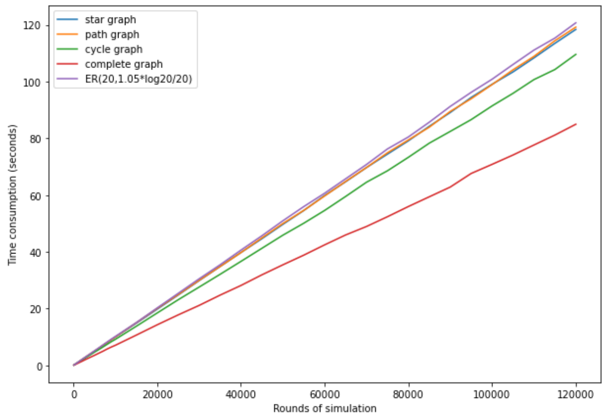

Figure 4.2: Time consumption for CMC simulations of different rounds

Figure 4.2 presents the time consumption for our crude Monte-Carlo simulator over different simulation rounds. We can see in the graph, as expected, the simulation time increases linearly with respect to the number of simulation rounds.

Figure 4.3, Figure 4.4, and Figure 4.5 present the precision performance of the CMC simulator over all p values. We can see that the error in complete graph simulation is very small compared to other graphs, this is what is expected due to all non-empty subgraphs of the complete graph is connected. Overall, the precision performance for different graph families are similar. It is noticed that in Figure 4.4, the relative error of path graph and ring(cycle) graph is higher than the other graph families. This is due to the long decreasing interval in the node reliablity of these two graphs shown in Figure 4.1.

With 10000 rounds of simulation, which only cost approximately 10 seconds from Figure 4.2, we can obtain an averaged error of 2e-3 over the entire interval of $p \in (0,1)$. With 70000 rounds of simulation, which would cost approximately 1 minute from Figure

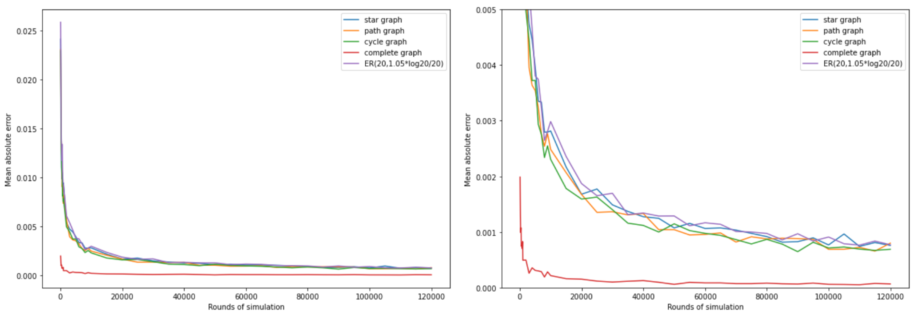

Figure 4.3: Mean absolute errors for CMC simulations of different rounds averaged over all p values

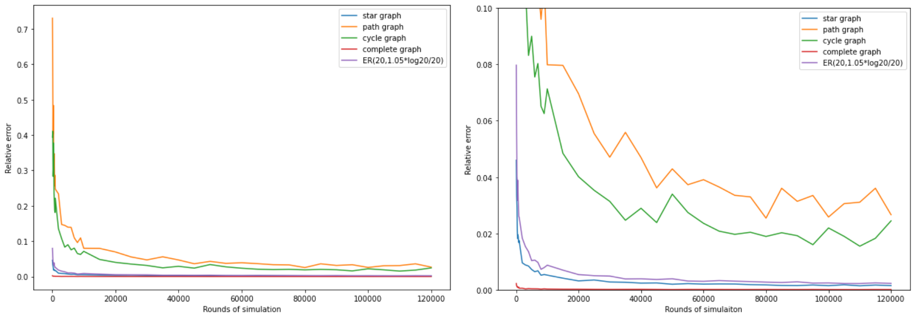

Figure 4.4: Relative errors for CMC simulations of different rounds averaged over all p values

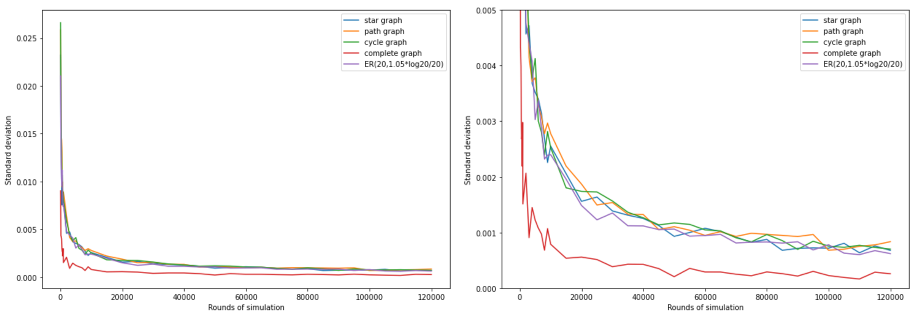

Figure 4.5: Standard deviation for CMC simulations of different rounds averaged over all p values

4.2, we can obtain an averaged error of 1e-3 over the entire interval of $p \in (0,1)$. This shows that it is indeed costly to improve the precision performance of a crude Monte-Carlo simulator.

It is also interesting to look at the performance of the crude Monte-Carlo simulator for different p values as the node reliability value ranges from zero to one. Due to the terrible performance of crude Monte-Carlo simulator over small rounds of simulations, we filter out the data whose rounds of simulation is smaller than 1e4 to obtain

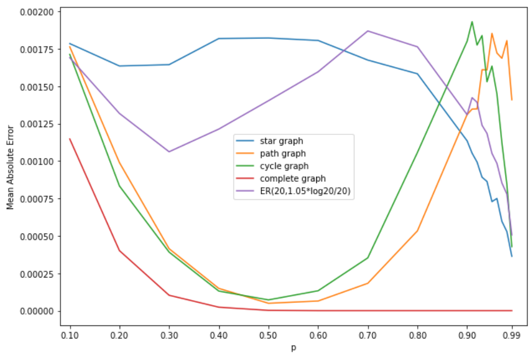

Figure 4.6: Mean absolute errors with respect to different p values

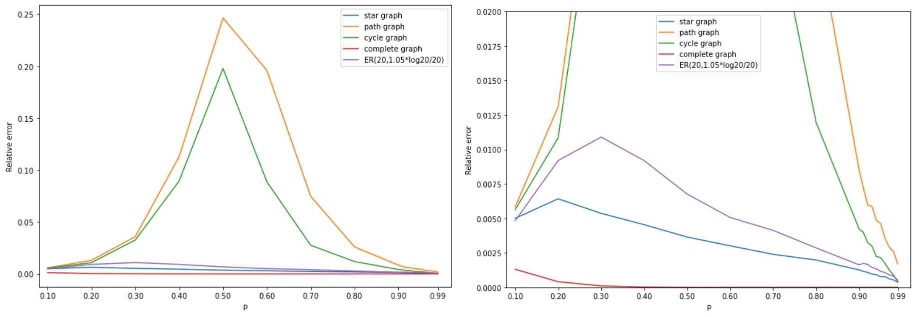

Figure 4.7: Relative errors with respect to different p values

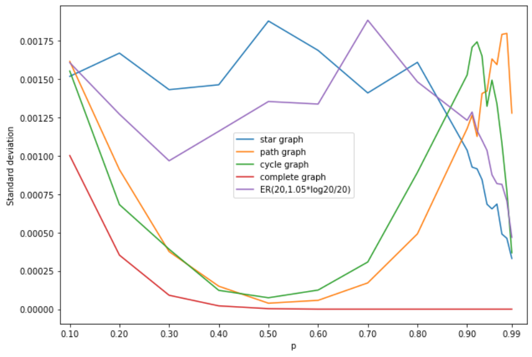

Figure 4.8: Standard deviation with respect to different p values

more meaningful results. Figure 4.6, Figure 4.7, and Figure 4.8 present the precision performance of the CMC simulator for different p values. As expected, the relative error is very high for path graph and ring graph for p values close to zero.

In fact, as we compare the true values and the errors of the CMC simulator, we can spot a reversed tendency. However, theoretically speaking, it is well know that CMC simulators are not so good at simulating rare events. According to the data we have obtained, for very unreliable cases, the graph remaining connected tends to become a rare event and indeed the error rises. But for very reliable cases, when the graph becoming disconnected tends to become a rare event, and the error does not rise. Especially for complete graphs, which is an extreme case that the graph becoming disconnected is almost impossible, but still has the lowest error in simulation. However, the complete graph is a very special construction, the node reliability polynomial values are too close to 1, such that the minor failure probability does not matter for precision thresholds that are normally accepted by researchers.

5 CONCLUSIONS

This last chapter summarizes our findings and suggests future research direction.

In this thesis, we have looked into four aspects of the properties of node reliability polynomials:

1. graph combination,
2. graph optimality,
3. intersection,
4. simulation.

In Chapter 2, we have presented the discovery and proofs of the additive properties of connected-sets in fully joint graphs and disjoint union graphs in Theorem 2.2 and 2.1. These properties have the potential to contribute to a node-reliability preserving graph reduction algorithm that reduces the computation load. However, the discovery of such an algorithm also relies on the further research on the graph partitioning problem, which, unfortunately, is generally a NP-hard problem. Besides, based on the optimality of the combinatorics of connected sets of different orders, we propose a conjecture that node-connectivity optimal complete bipartite graphs are optimal graphs. The analysis of this conjecture narrows the unproven problem to another conjecture that regular complete bipartite graphs $K_{n,n}$ are optimal graphs in class $\Omega(2n, n^2)$. Besides, we have shown that an upper bound of the node reliability of a graph can be derived when the degree sequence of the graph is known. Lastly, the exact solutions of some commonly used graph families are given in the last section of this chapter.

In Chapter 3, we have presented a construction method relying on the lexicographic operations to construct two graphs, with the same numbers of nodes and links, such that their node reliability polynomials intersect an arbitrary number of times. We further provided examples that contains graph pairs such that their node reliability polynomials intersection points range from 0 to 6, and one example with infinitely many intersections.

Chapter 4 presents the smallest graph sizes for graph pairs whose node reliability polynomials with 1, 2, and 3 intersections points. The performance of the crude Monte-Carlo simulation is discussed. It is shown that improving the precision of the crude Monte-Carlo simulator through increasing rounds of simulation is very costly indeed. And the simulator tends to perform worse when the true node reliability values are closer to zero, which often happens in large path graphs and cycle graphs. Overall, when the node operational probability gets closer to one in a connected graph, the precision performance of the crude Monte-Carlo simulator becomes better.

Still, there are many things unsolved and undiscovered in the field of node reliability polynomials. Here we propose some of the potential future work directions:

1. Are regular complete bipartite graphs $K_{n,n}$ optimal graphs?
2. An algorithm to reduce the computational load for node reliability polynomial is still unknown for general graphs or even for most graph families. It is interesting to see if, by any means, that there will be such an algorithm that would speed up the computations.
3. An efficient variance reduction technique for Monte-Carlo simulation of node reliability polynomial is still unknown. Most techniques and algorithms used in simulating all-terminal reliability are based on the coherency of the system, which, makes it impossible to apply them to the node reliability case.

BIBLIOGRAPHY

[1] B. Russell, *The Principles of Mathermatics*, 2nd ed. W. W. Norton Company, Feb. 1996, ch. Classes.

[2] J. van Heijenoort, *From Frege to Godel: A Source Book in Mathematical Logic 1879-1931*, ser. ISBN 978-0-674-32449-7. Harvard University Press, 1967, ch. An axiomatization of set theory, pp. 393–413.

[3] P. Van Mieghem, *Performance Analysis of Complex Networks and Systems.* Cambridge University Press, 2014.

[4] J. Brown and L. Mol, "The shape of node reliability," *Discrete Applied Mathematics*, vol. 238, pp. 41–55, Mar. 2018.

[5] Z. W. Birnbaum, J. D. Esary, and S. C. Saunders, "Multi-component systems and structures and their reliability," *Technometrics*, vol. 3, pp. 55–77, 1961.

[6] M. O. Ball, "Complexity of network reliability computations," *Networks*, vol. 10, no. 2, pp. 153–165, 1980.

[7] J. Carlier and C. Lucet, "A decomposition algorithm for network reliability evaluation," *Discrete Applied Mathematics*, vol. 65, no. 1, pp. 141–156, 1996.

[8] J. Silva, T. Gomes, D. Tipper, L. Martins, and V. Kounev, "An algorithm for computing all-terminal reliability bounds," in *Proc. 6th International Workshop on Reliable Networks Design and Modeling*, 2014, pp. 76–83.

[9] ——, "An effective algorithm for computing all-terminal reliability bounds," *Networks*, vol. 66, no. 4, pp. 282–195, 2015.

[10] A. Pönitz, "Über eine methode zur konstruktion von algorithmen fürdie berechnung von invarianten in endlichen ungerichtetenhypergraphen," PhD Thesis, Technical University Freiberg(Sachsen), 2003.

[11] A. M. Shooman, "Algorithms for network reliability and connection availability analysis," in *IEEE Electro/95 International Professional Program Proceedings*, 1995, p. 309–333.

[12] A. Satyanarayana and R. K. Wood, "A linear-time algorithm for computing k-terminal reliability in series-parallel networks," *SIAM Journal on Computing*, vol. 14, no. 4, pp. 818–832, 1985.

[13] H. Frank, "Maximally survivable node vulnerable networks," in *Memorandum for File, Div. Emergency Preparedness of Office of the President*, Washington, DC., Mar. 1969.

[14] J. Brown and L. Mol, “On the roots of the node reliability polynomial,” *Networks*, vol. 68, no. 3, pp. 238–246, 2016.

[15] J. I. Brown, C. J. Colbourn, D. Cox, C. Graves, and L. Mol, “Network reliability: Heading out on the highway,” *Networks*, vol. 77, no. 1, pp. 146–160, 2021.

[16] K. Sutner, A. Satyanarayana, and C. Suffel, “The complexity of the residual node connectedness reliability problem,” *SIAM Journal on Computing*, vol. 20, no. 1, pp. 149–155, 1991.

[17] J. Brown, “Maximal intervals of decrease and inflection points for node reliability,” *arXiv preprint arXiv:2103.13923*, Mar. 2021.

[18] C. J. Colbourn, A. Satyanarayana, C. Suffel, and K. Sutner, “Computing residual connectedness reliability for restricted networks,” vol. 44, pp. 221–232, 1993.

[19] L. Mol, “On connectedness and graph polynomials,” PhD Thesis, Dalhousie University, 2016.

[20] O. Goldschmidt, P. Jaillet, and R. Lasota, “On reliability of graphs with node failures,” *Networks*, vol. 24, no. 4, pp. 251–259, 1994.

[21] C. Stivaros, “On the residual node connectedness network reliaiblity model,” PhD Thesis, Stevens Institute of Technology, 1990.

[22] F. Boesch, A. Satyanarayana, and C. Suffel, “On residual connectedness network reliability,” pp. 51–60, 1991.

[23] S. Liu, K. Cheng, and X. Liu, “Network reliability with node failures,” *Networks: An International Journal*, vol. 35, no. 2, pp. 109–117, 2000.

[24] S. Yu, F.-M. Shao, and H. Meng, “Uniformly optimal graphs in some classes of graphs with node failures,” *Discrete mathematics*, vol. 310, no. 1, pp. 159–166, 2010.

[25] F. Boesch, D. Gross, and C. Suffel, “A coherent model for reliability of multiprocessor networks,” *IEEE Transactions on Reliability*, vol. 45, no. 4, pp. 678–684, 1996.

[26] R. M. Karp, “Reducibility among combinatorial problems,” in *Complexity of Computer Computations, The IBM Research Symposia Series.* New York: Plenum Press, 1972, p. 85–103.

[27] P. Van Mieghem, *Graph Spectra for Complex Networks.* Cambridge University Press, 2012, ch. 4.3 Partitioning of a Graph, pp. 89–96.

[28] J. I. Brown, Y. Koç, and R. E. Kooij, “Reliability polynomials crossing more than twice,” in *2011 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2011, pp. 1–6.

[29] W. Fulton, *Algebraic curves: an introduction to algebraic geometry.* Addison-Wesley, 1989, ch. Local Properties of Plane Curves, pp. 31–42.

[30] B. McKay and A. Piperno, “Practical graph isomorphism,” *Journal of Symbolic Computation*, vol. 60, no. 2, pp. 94–112, 2014.

A

Appendix: The complete python module for enumerating and simulating node reliability polynomials

```
# **** Documentation ****
#
# This python module contains several tools for users to
# compute Node reliability polynomials and relevant results.
#
# For more information concerning input output,
# look at the detailed description in the function code blocks.
#
# List of functions:
#
#
# ***Main functions***
#
# NRel
# ...Get the numerical number sequence of node reliability
# ...according to the input graph and probability sequence
#
# simulation
# ...Get the numerical number sequence of node reliability
# ...by Monte-Carlo simulation in user defined number of rounds
# ...according to the input graph and probability sequence
#
# NRelpolycoeff
# ...Get the (residual)node reliability polynomial coefficients
# ...Where the variable is p, the node operational probability
#
# NRelpolycoeff_F
# ...Get the (residual)node reliability polynomial coefficients
# ...Where the variable is q, the node failure probability
#
# connected_set_poly
# ...Get the node connected set polynomial coefficeints of a graph
#
# NRelpolycoeff_K_def1 (No longer updated!!!)
# ...Get the k-connected component node reliability polynomial
# ...(definition 1) of a graph
# ...
# ...Definition 1:
# ...Perfect links, nodes operates independently with probability p.
# ...At least k nodes operational, all operational nodes are connected.
#
# NRelpolycoeff_K_def1 (No longer updated!!!)
# ...Get the k-connected component node reliability polynomial
# ...(definition 2) of a graph
# ...
# ...Definition 2:
# ...Perfect links, nodes operates independently with probability p.
# ...At least k nodes operational,
# ...theres a connected subgraph of at least k nodes.
#
#
# ***Auxillery functions***
#
# combi
# combi_all
# PolySolve
# SecondDerivative
# FirstDerivative
# extrema_count
# inflection_count
# fixedpoint_count
# intersection_count
# examine_clusters
# load_adjacency
#
#*********************************************************************


import math
import statistics
import networkx as nx
import itertools as it
import numpy as np
import matplotlib.pyplot as plt
import random as rand
import scipy.special
import igraph as ig



def simulation(probability_seq,H,rounds):
    """
    Calculate the numerical result of the node reliability of a graph
    Input:
        p: node operation probability sequence, all values must belong to [0,1]
        H: a networkx graph or a list of tuples
            that describes the edges of a graph
        rounds: the number of rounds you want to simulate
    Output:
            The simulated numerical result of node reliability of the input graph
    """

    if type(H) is nx.classes.graph.Graph:
        G = ig.Graph.from_networkx(H)
    if type(H) is ig.Graph:
        G = H.copy()

    kappa = G.cohesion()
    N = G.vcount()
    result = np.zeros(len(probability_seq))
    for i in range(rounds):
        for k,pp in enumerate(probability_seq):
            states = np.random.choice([0, 1], size=(N,), p=[1-pp, pp])
            failed_nodes = np.array(np.where(states==0))[0]
            if len(failed_nodes) < kappa:
                result[k] += 1/rounds
            else:
                GG = G.copy()
                GG.delete_vertices(failed_nodes)
                if len(failed_nodes) != N and GG.is_connected() == True:
                    result[k] += 1/rounds
    return result

def polyplot(R):
    x = np.linspace(0,1,100)
    plt.plot(x,PolySolve(x,R))

def combi(N,num):
    """
        Find all comibinations of 'num' elements for int in [0,N-1].

        Input:
            N total number of elements (numbering starts from zero!)
            num number of elements in one combination

        Output:
            C the list with all combinations
    """
    return tuple(it.combinations(tuple(range(N)),num))


def combi_all(N):
    """
    Find all comibinations for int in [0,N-1].

    Input:
        N total number of elements (numbering starts from zero!)

    Output:
        AC the list with all combinations,
            arranged from the empty set to N-element sets
    """
    return tuple(map(combi,it.repeat(N),range(N+1)))


def connected_set_poly(H):
    """
    This function generates the connected set node polynomial of a graph
    Input:
        H H can be a list of tuples that describes all edges in a graph,
              or a networkx graph.
    Output:
        s The node connected set polynomial coefficients,
              begins with the leading coefficient and ends
              with the constant coefficient
    """

    if type(H) is nx.classes.graph.Graph:
        G0 = H.copy()
        N = G0.order()
        mapping = dict(zip(G0, range(N)))
        G0 = nx.relabel_nodes(G0,mapping)
        G = ig.Graph.from_networkx(G0)
    elif type(H) is ig.Graph:
        G = H.copy()
        N = G.vcount()
    else:
        N = len(np.unique(H)) #number of nodes
    # G = nx.Graph(H)
        G = ig.Graph(N,H)

    s=[0]*(N+1)
    if N>0:
        s[1] = N
        if N>1:
            s[2] = G.ecount()
            kappa_det = G.cohesion()
            for i in range(kappa_det):
                s[N-i] = math.comb(N,N-i)
            AC = []
            for i in range(kappa_det,N+1-3):
                AC.extend(combi(N,i))
            for t in AC:
                GG = G.copy()
                GG.delete_vertices(t)
                if GG.is_connected() == True:
                    l = N-len(t)
                    s[l] += 1
    s.reverse()
    return s

def load_adjacency(filename):


    g = open(filename,'r')
    lines = g.readlines()
    lines = [x for x in lines if x!='\n']
    G = []
    A = []
    for i in range(len(lines)):
        if 'Graph' in lines[i]:
            lines[i] = 'G'+lines[i][lines[i].index(' ')+1:lines[i].index(',')]
            if i != 0:
                G.append(nx.Graph(np.array(A)))
                A = []
        else:
            lines[i] = lines[i][0:lines[i].index('\n')]
            A.append(list(map(int,lines[i].split(' '))))
    G.append(nx.Graph(np.array(A)))

    return G

def NRel(p,H):
    """
    Calculate the numerical result of the node reliability of a graph
    Input:
        p node operation probability sequence, all values must belong to [0,1]
        H a networkx graph or a list of tuples that describes the edges of a graph
    Output:
            The numerical result of node reliability of the input graph
```

```python
    """

    if type(H) is nx.classes.graph.Graph:
        G0 = H.copy()
        N = G0.order()
        mapping = dict(zip(G0, range(N)))
        G0 = nx.relabel_nodes(G0,mapping)
        G = ig.Graph.from_networkx(G0)
    elif type(H) is ig.Graph:
        G = H.copy()
        N = G.vcount()
    else:
        N = len(np.unique(H)) #number of nodes
    # G = nx.Graph(H)
        G = ig.Graph(N,H)

    s = connected_set_poly(H)
    s.reverse()
    R_tot = [0]*len(p)
    for j,q in enumerate(p):
        R_temp = 0
        for i,ss in enumerate(s):
            R_temp += ss*(q**i)*((1-q)**(N-i))
        R_tot[j] = R_temp
    return R_tot


def get_subsets(fullset):
    """
        This function gives the power set of of link; the size of power set is 2^L
    """
    listrep = list(fullset)
    n = len(listrep)
    return [[listrep[k] for k in range(n) if i & 1 << k] for i in range(2 ** n)]


def NRelpolycoeff(H):
    """
    This function generates the (residual)node reliability polynomial of a graph
    With p --- the node operation probability, as the variable of the polynomial
    Input:
        H H can be a list of tuples that describes all edges in a graph,
            or a networkx graph.
    Output:
        R The (residual)node reliability polynomial coefficients,
            with p --- the node operation probability,
            as the variable of the polynomial begins with the
            leading coefficient and ends with the constant coefficient
    """
    if type(H) is nx.classes.graph.Graph:
        G0 = H.copy()
        N = G0.order()
        mapping = dict(zip(G0, range(N)))
        G0 = nx.relabel_nodes(G0,mapping)
        G = ig.Graph.from_networkx(G0)
    elif type(H) is ig.Graph:
        G = H.copy()
        N = G.vcount()
    else:
        N = len(np.unique(H)) #number of nodes
    # G = nx.Graph(H)
        G = ig.Graph(N,H)

    s = connected_set_poly(G)
    s.reverse()
    s = tuple(s)
    R = [0]*(N+1)

    for k in range(N+1):
        f = N-k
        r_temp = [0]*(f+1)
        for j in range(f+1):
            r_temp[j] = (-1)**(j)*s[k]*math.comb(f, j)
            R[k+j] += r_temp[j]
    R.reverse()
    # s.reverse()
    return R


def PolySolve(x, coeffs):
    """
    Gives the numerical result of the polynomials
    corresponding to the coefficent sequence of x.
    Input:
        x numerical sequence of the variable
        coeffs coefficients of that polynomial
                (starts from the leading coefficient)
    Output:
        y The numerical result of the coefficient sequence
            corresponding to the input numerical sequence of the variable
    """
    y = 0
    for i in range(len(coeffs)):
        y += coeffs[-i-1]*x**i
    return y


def FirstDerivative(R):
    """
    Find the coefficients of the first derivative of a polynomial function
    The polynomial coefficients start from the leading
    coefficient to the constant coefficient.
    Input:
        R The input polynomial coefficient sequence
    Output:
        r The polynomial coefficient sequenceof the first
            derivative of the input polynomial R
    """
    RR = R.copy()
    l = len(RR)
    r = [0]*(l-1)
    for i in range(len(r)):
        power = l-i-1
        r[i] = RR[i] * power
    return r


def SecondDerivative(R):
    """
    Find the coefficients of the second derivative of a polynomial function
    The polynomial coefficients start from the
    leading coefficient to the constant coefficient.
    Input:
        R The input polynomial coefficient sequence
    Output:
        r The polynomial coefficient sequence
            second derivative of the input polynomial R
    """
    RR = R.copy()
    l = len(RR)
    r = [0]*(len(RR)-2)
    for i in range(len(r)):
        power = l-i-1
        r[i] = power*(power-1)*R[i]

    return r


def FindnumofRootsZeros2One(y):
    """
    This function finds the number of roots in (0,1)
    ***Not used anymore***
    """
    x = np.linspace(0, 1, 10000)
    count = 0
    minus = 0
    yy = y.copy()
    del yy[-1]
    for i in range(len(yy)):
        if i == 0 or i == 1:
            continue
        else:
            if yy[i]*yy[i-1] <=0:
                count += 1
            if yy[i] == 0:
                minus += 1
    count = count - minus
    return count


def extrema_count(R,p):
    """
    Statistics of the number of extemas of the input polynomials.
    Input:
        R List of polynomial coefficient lists.
            The polynomial coefficients should start from leading
            coefficient and ends with the constant coefficient.
        p Print flag. Iff p is set to 1, then print the result, else no print.
    Output:
        cc The list contains the statistics of the
            number of extremas of the input polynomials.
    """
    C = []
    for r in R:
        r1 = FirstDerivative(r)
        c1 = len(list(set([np.round(np.real(x),4) for x in list(np.roots(r1))
                if (np.isreal(x) == 1 and np.real(x)>0 and np.real(x)<0.99)])))
        C.append(c1)
    cc = [0,0,0,0,0,0]
    for i in range(len(C)):
        for k in range(len(cc)):
            if C[i] == k:
                cc[k] += 1
    if p == 1:
        print('number of extremas')
        print(' 0 1 2 3 4 5')
        print(cc)
    return cc


def inflection_count(R,p):
    """
    Statistics of the number of inflection points of the input polynomials.
    Input:
        R List of polynomial coefficient lists.
            The polynomial coefficients should start from leading
            coefficient and ends with the constant coefficient.
        p Print flag. Iff p is set to 1, then print the result, else no print.
    Output:
        cc The list contains the statistics of the number of
            inflection points of the input polynomials.
    """
    C = []
    for r in R:
        r1 = SecondDerivative(r)
        c1 = len(list(set([np.round(np.real(x),4) for x in list(np.roots(r1))
                  if (np.isreal(x) == 1 and np.real(x)>0 and np.real(x)<0.99)])))
        C.append(c1)
    cc = [0,0,0,0,0,0]
    for i in range(len(C)):
        for k in range(len(cc)):
            if C[i] == k:
                cc[k] += 1
    if p == 1:
        print('number of inflection points')
        print(' 0 1 2 3 4 5')
        print(cc)
    return cc


def fixedpoint_count(R,p):
    """
    Statistics of the number of fixed points of the input polynomials.
    Input:
        R List of polynomial coefficient lists.
            The polynomial coefficients should start from leading
            coefficient and ends with the constant coefficient.
        p Print flag. Iff p is set to 1, then print the result, else no print.
    Output:
        cc The list contains the statistics of the
            number of fixed points of the input polynomials.
    """
    step_size = 0.05
    p0 = np.arange(0,1+step_size,step_size).tolist()
    p = [round(num,2) for num in p0]
    Cf = []
    x = np.linspace(0,1,100)
    for r in R:
        rrr = r.copy()
        rrr[-2] -= 1
        c = len(list(set([np.round(np.real(x),4) for x in list(np.roots(rrr))
                    if (np.isreal(x) == 1 and np.real(x)>0 and np.real(x)<0.9999)])))

        Cf.append(c)

    cc = [0,0,0,0,0,0]
    for i in range(len(Cf)):
        for k in range(len(cc)):
            if Cf[i] == k:
                cc[k] += 1
                continue
    if p == 1:
        print('number of fixed points')
        print(' 0 1 2 3 4 5')
        print(cc)
    return cc


def intersection_count(R,p):
    """
    Statistics of the number of intersection points of all possible
    2-combinations of the input polynomials.
    Input:
        R List of polynomial coefficient lists.
            The polynomial coefficients should start from
            leading coefficient and ends with the constant coefficient.
        p Print flag. Iff p is set to 1, then print the result, else no print.
    Output:
        cc The list contains the statistics of the number of
        intersection points of all possible 2-combinations of the input polynomials.
    """
    step_size = 0.05
    Ci = []
    for i in range(len(R)):
        for j in range(len(R)):
            if i != j:
                r1 = R[i].copy()
                r2 = R[j].copy()
                r0 = [r1[x]-r2[x] for x in range(len(r1))]
                c = len(list(set([np.round(np.real(x),4) for x in list(np.roots(r0))
                            if (np.isreal(x) == 1 and np.real(x)>0 and np.real(x)<0.9999)])))
                Ci.append(c)

    cc = [0,0,0,0,0,0]
    for i in range(len(Ci)):
        for k in range(len(cc)):
            if Ci[i] == k:
                cc[k] += 1
    ccc = [int(i/2) for i in cc]
    if p == 1:
        print('number of intersection points')
        print(' 0 1 2 3 4 5')
        print(ccc)
    return ccc
```

```


def examine_clusters(selection,k,G,pr):
    """
    Input:
        selection: select what to examine,
           1 is the node reliability polynomial
                with node operational rate p
           2 is the node reliability polynomial
                 with node failure rate q
           3 is the K-node operating componenet realibility
                (definition 1*)
           4 is the K-node operating componenet realibility
                (definition 2**)
        k: a list of k values to be examined
        G: a list of networkx graphs
        pr: print the result if pr equals to 1.

    Output:
        E: a list of the statistics***
           of the number of extremas of each graph
        I: a list of the statistics***
           of the number of inflection points of each graph
        F: a list of the statistics***
           of the number of fixed points of each graph
        IS: a list of the statistics***
           of the number of intersection points
           of all possible graph combinations


    *Definition 1: Perfect links, nodes operates
                   independently with probability p.
                   At least k nodes operational,
                   all operational nodes are connected.

    **Definition 2: Perfect links, nodes operates
                    independently with probability p.
                    At least k nodes operational,
                    there's a connected subgraph of at least k nodes.

    ***statistics: Currently a naive way of counting is used,
                   the counting starts from 0 and reaches 5.

                   If the output list is [10,20,30,40,50,60] for E(extremas),
                   this means there are 10 graphs with 0 extrema;
                   there are 20 graphs with 1 extremas;
                   there are 30 graphs with 2 extremas;
                   there are 40 graphs with 3 extremas;
                   ......
```

```

                  The same goes for all E, I, F, IS.
    """
    E = []
    I = []
    F = []
    IS = []
    for x in k:
        print(x)
        R_t = []
        for i in G:
            if selection == 1:
                R = NRelpolycoeff(list(i.edges()))
            if selection == 2:
                R = NRelpolycoeff_F(list(i.edges()))
            if selection == 3:
                R = NRelpolycoeff_K_def1(list(i.edges()),x)
            if selection == 4:
                R = NRelpolycoeff_K_def2(list(i.edges()),x)
            R_t.append(R)

        E.append(extrema_count(R_t,pr))
        I.append(inflection_count(R_t,pr))
        F.append(fixedpoint_count(R_t,pr))
        IS.append(intersection_count(R_t,pr))
    return E,I,F,IS









def NRelpolycoeff_F(H):
    """
    This function generates the (residual)node reliability polynomial
    of a graph with q = 1-p --- the node failure probability,
    as the variable of the polynomial
    Input:
        H H can be a list of tuples that describes all edges in a graph,
                or a networkx graph.
    Output:
        R The (residual)node reliability polynomial coefficients,
                with q = 1-p --- the node failure probability,
                as the variable of the polynomial
                begins with the leading coefficient
                and ends with the constant coefficient
    """

    if type(H) is nx.classes.graph.Graph:
        G = H.copy()
        N = G.order()
        mapping = dict(zip(G, range(N)))
        G = nx.relabel_nodes(G,mapping)
    else:
        N = len(np.unique(H)) #number of nodes
        G = nx.Graph(H)
    s = connected_set_poly(H)
    s.reverse()
    R = [0]*(N+1)
    for k in range(N+1):
        f = N-k
        r_temp = [0]*(k+1)
        for j in range(k+1):
            r_temp[j] = (-1)**(j)*s[k]*scipy.special.comb(k, j, exact=True)
            R[j+f] += r_temp[j]
    R.reverse()
    s.reverse()
    return R


def NRelpolycoeff_K_def1(H,k):
    """
    This function generates the k-connected
    component node reliability polynomial
    (definition 1***)of a graph
    With p --- the node operation probability,
    as the variable of the polynomial
    Input:
        H H can be a list of tuples that describes
            all edges in a graph.
    Output:
        R The k-connected component node reliability
            polynomial coefficients, with p --- the node
            operation probability, as the variable of the polynomial
            begins with the leading coefficient and
            ends with the constant coefficient



    ***Definition 1: Perfect links,
                   nodes operates independently with probability p.
                   At least k nodes operational,
                   all operational nodes are connected.
    """

    if type(H) is nx.classes.graph.Graph:
        G = H.copy()
        N = G.order()
        mapping = dict(zip(G, range(N)))
        G = nx.relabel_nodes(G,mapping)
    else:
        N = len(np.unique(H)) #number of nodes
        G = nx.Graph(H)
    s = connected_set_poly(H)
    addition = [0]*k
    s0 = [s[i] for i in range(N-k+1)]
    s0.extend(addition)
    s = s0.copy()
    s.reverse()
    R = [0]*(N+1)

    for k in range(N+1):
        f = N-k
        r_temp = [0]*(f+1)
        for j in range(f+1):
            r_temp[j] = (-1)**(j)*s[k]*scipy.special.comb(f, j, exact=True)
            R[k+j] += r_temp[j]
    R.reverse()
    s.reverse()
    return R



def NRelpolycoeff_K_def2(H,k):
    """
    This function generates the k-connected
    component node reliability polynomial
    (definition 2***)of a graph
    With p --- the node operation probability,
    as the variable of the polynomial
    Input:
        H H can be a list of tuples
                that describes all edges in a graph.
    Output:
        R The k-connected component node reliability
                 polynomial coefficients,
                with p --- the node operation probability,
                as the variable of the polynomial
                begins with the leading coefficient
                and ends with the constant coefficient



    ***Definition 2: Perfect links,
                    nodes operates independently with probability p.
                    At least k nodes operational,
                    there's a connected subgraph of at least k nodes.

    """

    if type(H) is nx.classes.graph.Graph:
        G = H.copy()
        N = G.order()
        mapping = dict(zip(G, range(N)))
        G = nx.relabel_nodes(G,mapping)
    else:
        N = len(np.unique(H)) #number of nodes
        G = nx.Graph(H)
    s = connected_set_poly(H)
    s.reverse()
    R = [0]*(N+1)

    for k in range(N+1):
        f = N-k
        r_temp = [0]*(f+1)
        for j in range(f+1):
            r_temp[j] = (-1)**(j)*s[k]*scipy.special.comb(f, j, exact=True)
            R[k+j] += r_temp[j]
    R.reverse()
    s.reverse()
    return R


def AllTerminalReliablility(p,edgelist):
    N = len(list(np.unique(edgelist)))+1
    M = len(edgelist)
    all_states = combi_all(M)
    R_tot = []
    G = nx.Graph(edgelist)

    for q in p:
        R = 0
        for i in range(len(all_states)):
            GG = G.copy()
            rem = [edgelist[x] for x in all_states[i]]
            GG.remove_edges_from(rem)
            if nx.is_connected(GG):
                k = len(all_states[i])
                x_k = (q**(M-k)) * ((1-q)**(k))
                R = R + x_k
        R_tot.append(R)
    return R_tot

def allterminalpolycoeff(H):
    L = len(H) #number of links
    P = list(get_subsets(set(H)))
    print(len(P))
    N = len(np.unique(H)) #number of nodes

    GG = nx.empty_graph(N)
    GG.add_edges_from(H)
    nx.draw(GG,with_labels = True,node_color='yellow')
    plt.show()

    s=[0]*(L+1)


    for t in range(2 ** L):
        G = nx.empty_graph(N)
        l = len(P[t])
        G.add_edges_from(P[t])
        Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
        #ordered list containing all connected components
        G0 = Gcc[0] #size of largest connected component
        if (len(G0) == N):
            s[l] += 1
    R=[0]*(L+1)
    for k in range(L+1):
      som=0
      for j in range(k+1):
        som=som+(-1)**(j+k)*s[j]*scipy.special.comb(L-j, L-k, exact=True)
      R[k]=som
    R.reverse()
    return R
```