

Southern Methodist University

SMU Scholar

Computer Science and Engineering Theses and
Dissertations

Computer Science and Engineering

Spring 5-13-2023

Visualized Algorithm Engineering on Two Graph Partitioning Problems

Zizhen Chen

Southern Methodist University, zizhenc@smu.edu

Follow this and additional works at: https://scholar.smu.edu/engineering_compsci_etds



Part of the [Graphics and Human Computer Interfaces Commons](#), [Numerical Analysis and Scientific Computing Commons](#), [Programming Languages and Compilers Commons](#), [Software Engineering Commons](#), and the [Theory and Algorithms Commons](#)

Recommended Citation

Chen, Zizhen, "Visualized Algorithm Engineering on Two Graph Partitioning Problems" (2023). *Computer Science and Engineering Theses and Dissertations*. 29.
https://scholar.smu.edu/engineering_compsci_etds/29

This Dissertation is brought to you for free and open access by the Computer Science and Engineering at SMU Scholar. It has been accepted for inclusion in Computer Science and Engineering Theses and Dissertations by an authorized administrator of SMU Scholar. For more information, please visit <http://digitalrepository.smu.edu>.

VISUALIZED ALGORITHM ENGINEERING
ON
TWO GRAPH PARTITIONING PROBLEMS

Approved by:

Dr. David W. Matula
Professor Emeritus of Computer Science

Dr. Ira Greenberg
Professor and Director Center of Creative
Computation

Dr. Eli Olinick
Associate Professor of Operations
Research and Engineering Management

Dr. Michael Hahsler
Clinical Associate Professor of Computer
Science

Dr. Jennifer Dworak
Associate Professor of Electrical and
Computer Engineering

Dr. Frank Coyle
Director of Software Engineering Program

VISUALIZED ALGORITHM ENGINEERING
ON
TWO GRAPH PARTITIONING PROBLEMS

A Dissertation Presented to the Graduate Faculty of the

Bobby B. Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Science

by

Zizhen Chen

M.S., Computer Science, Southern Methodist University
B.SE., Software Engineering, North China Electric Power University

May 13, 2023

Copyright (2023)

Zizhen Chen

All Rights Reserved

ACKNOWLEDGMENTS

This work could not have been accomplished without guidance from my thesis advisor, Prof. David W. Matula who provides key ideas and forever support. I want to thank all faculties and students from both Computer Science and Art departments as the original idea of this work was hatched from both environments. Special appreciation goes to Prof. Ira Greenberg, who provided a lot of insights on visualization and also gave me opportunities to found my teaching skills. I'm also grateful to my family for being so patient with me and my friends for reminding me to take breaks instead of working all day.

Chen, Zizhen

M.S., Computer Science, Southern Methodist University
B.SE., Software Engineering, North China Electric Power University

Visualized Algorithm Engineering
on
Two Graph Partitioning Problems

Advisor: Dr. David W. Matula

Doctor of Philosophy degree conferred May 13, 2023

Dissertation completed March 24, 2023

Concepts of graph theory are frequently used by computer scientists as abstractions when modeling a problem [21]. Partitioning a graph (or a network [28]) into smaller parts is one of the fundamental algorithmic operations that plays a key role in classifying and clustering. Since the early 1970s, graph partitioning rapidly expanded for applications in wide areas. It applies in both engineering applications, as well as research [8]. Current technology generates massive data (“Big Data”) from business interactions and social exchanges [11, 66], so high-performance algorithms of partitioning graphs are a critical need.

This dissertation presents engineering models for two graph partitioning problems arising from completely different applications, computer networks and arithmetic. The design, analysis, implementation, optimization, and experimental evaluation of these models employ visualization in all aspects. Visualization indicates the performance of the implementation of each Algorithm Engineering work, and also helps to analyze and explore new algorithms to solve the problems. We term this research method as “Visualized Algorithm Engineering (VAE)” to emphasize the contribution of the visualizations in these works.

The techniques discussed here apply to a broad area of problems: computer networks, social networks, arithmetic, computer graphics and software engineering. Common terminologies accepted across these disciplines have been used in this dissertation to guarantee practitioners from all fields can understand the concepts we introduce.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	xi
LIST OF VIDEOS	xiii
CHAPTER	
1. INTRODUCTION	1
1.1. Problem Specifications	1
1.1.1. Backbone Determination in Wireless Sensor Networks (WSNs)	2
1.1.2. Graphical Partitioning of the Natural Number Network	2
1.2. Software Environments	3
1.3. Outline of The Thesis	4
1.4. Contributions	4
1.4.1. Backbone Determination in Wireless Sensor Networks (WSNs)	4
1.4.2. Graphical Partitioning of the Natural Number Network	5
1.4.3. Visualized Algorithm Engineering (VAE)	6
2. BACKBONE DETERMINATION IN WIRELESS SENSOR NETWORKS.....	7
2.1. Introduction	7
2.2. Background	9
2.2.1. Network Computational Model	9
2.2.2. Cluster-based Formation	10
2.3. Backbone Formation	11
2.3.1. Primary and Relay Bipartite Set	12
2.3.2. Backbone Determination via Multi-partitioning	14
2.4. Visualized Algorithm Engineering.....	16
2.4.1. Sensor Deployment.....	16

2.4.2.	Link Determination	21
2.4.2.1.	Sweep Method	22
2.4.2.2.	Cell Method	23
2.4.3.	Backbone Partitioning	28
2.4.3.1.	Smallest-last Coloring	31
2.4.3.2.	Relay Coloring	40
2.4.4.	Backbone Refinement	46
2.4.4.1.	Performance Metrics	49
2.4.4.2.	Robustness of Backbones	51
2.4.4.3.	Other Adjustments	56
2.5.	Conclusion	60
2.5.1.	Contributions of Visualized Algorithm Engineering (VAE)	62
2.5.2.	Evaluation and Future Works	63
3.	GRAPHICAL PARTITIONING OF THE NATURAL NUMBER NETWORK ..	66
3.1.	Introduction	66
3.2.	Background	67
3.3.	Graphical Representation of Counts	69
3.3.1.	Font Representation System	69
3.3.2.	Multiplicative Properties of the Graphical Representation of Counts	73
3.3.3.	Evaluations of the Graphical Number Representation	76
3.4.	Adjacency Relation Between Natural Numbers	77
3.4.1.	Non-isomorphic trees of Matula numbers	77
3.4.2.	Specifications of Primordial Trees	79
3.5.	Visualized Algorithm Engineering	81
3.5.1.	Algorithms of Prime Functions	81
3.5.1.1.	Prime Factorization	82

3.5.1.2.	Prime Counting and k_{th} Prime	83
3.5.2.	Matula Number Generator	85
3.5.3.	Primordial Tree Generator	88
3.5.3.1.	Brute Force Method of Primordial Tree Construction	88
3.5.3.2.	Integer Connectivity	89
3.5.3.3.	Primordial Tree Construction via Integer Connectivity	91
3.5.4.	$i \cdot p_j$ Matrix.....	94
3.5.5.	Primordial Spiral.....	95
3.6.	Conclusion	100
3.6.1.	Contributions of Visualized Algorithm Engineering (VAE).....	101
3.6.2.	Future Works	101
4.	VISUALIZATION'S ROLE IN ALGORITHM ENGINEERING.....	102
4.1.	Introduction to Visualized Algorithm Engineering (VAE)	102
4.2.	Evaluate VAE on the Two Graph Partitioning Problems	102
4.3.	Creative Coding.....	103
4.4.	Conclusion	104
APPENDIX		
BIBLIOGRAPHY		105

LIST OF FIGURES

Figure	Page
2.1 Wireless Sensor Network	8
2.2 Manually Placed Bipartite Lattice Grids	13
2.3 Triangular Lattice Independent Sets	13
2.4 Coverages of Primary Set and Relay Set	14
2.5 Bi-regular Three and Four Lattice Grid	14
2.6 Models of Four Geometries	17
2.7 Random Nodes in Unit Disk	18
2.8 Ratio of Areas and Nodes	20
2.9 Degree Estimation of Nodes in Border Area	20
2.10 Random Node Distribution on Four Surfaces	21
2.11 Sweep Method on 2D Coordinate Systems	22
2.12 Sweep Method on 3D Coordinate Systems	23
2.13 Cell Method on 2D Coordinate Systems	26
2.14 Cell Method on 3D Coordinate Systems	27
2.15 Unit Sphere $G(6400, 0.251)$ Cell Method	28
2.16 Smallest-last Ordering	31
2.17 Degree List	32
2.18 Smallest-last Ordering	33
2.19 Node-degree Plots of Smallest-last Ordering	34
2.20 Smallest-last Coloring	36
2.21 Color Size Plot of Smallest-last Coloring	37

2.22	Degree-3 Faces of Selected Primary Color Sets with Gabriel Rules	38
2.23	Random-paired Selected Bipartite Subgraphs	39
2.24	Degree-3 Faces Percentage Plot	40
2.25	Relay Degree List	41
2.26	Maximum One-color Neighbors	41
2.27	Color Degree List	42
2.28	Relay Coloring Procedure	43
2.29	Color Size Plot of Relay Coloring	45
2.30	Degree-3 Faces of Selected Relay Color Sets with Gabriel Rules	45
2.31	Nodes after Smallest-last and Relay Coloring	46
2.32	Degree-3 Face Percentage of Primary and Relay Color Sets	48
2.33	Sample Paired Bipartite Subgraphs	49
2.34	Sample Bipartites with Minor Components	51
2.35	Domination and 3-coverage of Sample Backbones of Different Types	57
2.36	Number of Backbones of k Selected Primary Sets	58
2.37	Surplus Nodes vs. Selected Primary Nodes	59
2.38	Performance of k Selected Primary Colors	59
2.39	Performances of Different Connectivities	60
2.40	Surplus Nodes of Different Connectivities	61
2.41	Sample Backbones of Different Connectivities	61
2.42	Sample Backbones of Other Geometries	62
3.1	21 Coefficients of Continued Fraction of π	67
3.2	Graphic Representations of Natural Number 292	70
3.3	Comparison of Liquid-Crystal Display (LCD) Font and Grid Representations of Numbers 1~11	71
3.4	Initial 21 Coefficients of Continued Fraction of π	72

3.5	The Structure of the Rooted Tree of Matula Number 20.....	77
3.6	Non-isomorphic Tree of Number Set $\{20, 21, 29, 34, 59\}$	78
3.7	The Sieve of Eratosthenes Applied on Numbers Up to 20.....	84
3.8	Adjacency List with Edges	86
3.9	Primordial Tree t_{20}	91
3.10	Observe the Exterior from within the Primordial Constellation	93
3.11	20×20 $i \cdot p_j$ Matrix S_{20}	94
3.12	$i \cdot p_j = j \cdot p_i$ Cells of S_{20}	96
3.13	Nodes of Primordial Tree t_{20}	96
3.14	Prime Spiral	97
3.15	Number Spiral	98
3.16	Primordial Spiral	98
3.17	Prime and Centrum Numbers of the Number Spiral	99
3.18	Primordial Trees: t_2, t_3, t_5, t_7	99
4.1	Domination of Sample Backbones	104

LIST OF TABLES

Table	Page
2.1 Definitions or Formulas of Geometries	19
2.2 Unit Square Sweep Method.....	24
2.3 Unit Disk Sweep Method	24
2.4 Unit Sphere Sweep Method.....	25
2.5 Unit Torus Sweep Method.....	25
2.6 Unit Square Cell Method.....	28
2.7 Unit Disk Cell Method	29
2.8 Unit Sphere Cell Method.....	29
2.9 Unit Torus Cell Method	30
2.10 Smallest-last Ordering.....	34
2.11 Smallest-last Coloring Procedure	37
2.12 Relay Coloring	44
2.13 Nodes after Smallest-last and Relay Coloring	47
2.14 Evenly Selected Sample Backbones	50
2.15 Domination and 3-coverage Percentage	52
2.16 Sample Backbones of Different Robustness.....	54
3.1 Number 1~9 in Matula Number and Grid Graphic Representation	70
3.2 Compact Grid Symbols for Numbers of Automorphism Groups	71
3.3 Compact Grid Symbols for Series of Repeating Numbers	72
3.4 Multiplicative Operations	74
3.4 Multiplicative Operations, continued.	75

3.5	The Rooted Trees of Matula Number Set $\{20, 21, 29, 34, 59\}$	78
3.6	Examples of Primordial Trees with Repeating Numbers	81

LIST OF VIDEOS

Video	Page
2.1 Sensor Deployment	21
2.2 Link Determination	30
2.3 RGG Degeneracy	35
2.4 Smallest-last Coloring	39
2.5 Relay Coloring	48
2.6 Backbone Refinement	56
3.1 Matula Number 292	69
3.2 Primordial Tree t_{147}	90
3.3 Primordial Constellation	93
3.4 Primordial Garden	100

This is dedicated to my wife and son (Mengnan and Derek) who have been constant sources of support and encouragement during the challenges of graduation.

Chapter 1

INTRODUCTION

About 1/3 of the cerebral cortex is used for visual processing in our brain [33], so humans are basically a vision-driven species. “What does it look like?” is probably the most straightforward question popping up when we try to know about a new thing. The use of visualization helps to answer the question and has existed for thousands of years in human history. You must have impressions of old maps appearing in many adventurous fairy tails. An early example of using visualization in scientific research is the “seven bridges of Königsberg” problem. In 1763, Leonhard Euler proved his conclusion of the problem by drawing lands as nodes and bridges as edges [8]. This modeling method has been commonly recognized as the beginning of Graph Theory.

Algorithms are a fascinating use case for visualization. Instead of merely plotting data to charts (as what data visualization usually does), algorithm visualization is an animation process that describes the behavior with logical rules. This facilitates the analysis and understanding of abstract processes of algorithms via leveraging the human visual system.

This dissertation presents algorithm engineering works of two separate problems,

- Backbone Determination in Wireless Sensor Networks (WSNs);
- Graphical Partitioning of the Natural Number Network.

Although the two problems are cases across completely different areas including computer networks and arithmetics, both of them are modeled as graph partitioning problems. The engineering works here are all in visual forms which termed Visualized Algorithm Engineering (VAE).

1.1. Problem Specifications

1.1.1. Backbone Determination in Wireless Sensor Networks (WSNs)

WSNs have been the focus of intense research during the past few years because of their potential to facilitate data acquisition and scientific studies [73]. Lack of a fixed infrastructure and dynamic network topology make the routing problem one of the most challenging issues in the WSN area. Flooding is one solution in most topology-based routing protocols, but it suffers from the notorious broadcast storm problem and is extremely resource consuming [68]. Forming a virtual backbone that forwards the packets along it is a more cost-efficient method compared with the flooding approach.

We investigate the method of backbone determination by partitioning a WSN into a limited number of densely packed disjoint bipartite grids. The study focuses on the collection of grids each individually having similar size and structure, and the union employing most of the sensor nodes. The residual nodes we seek to minimize are attributed to the inherent variations in densities of the randomly placed nodes and to any shortcomings of our greedy algorithms. Computational visual results are rendered on WSNs of different graphic models (unit square, unit disk, unit sphere and unit torus). Our results are documented by performance metrics of WSNs' such as coverage and network lifetime. The applications and future works of our backbone formation method in both centralized and distributed approaches are also discussed. This content is based on our former published paper [17] with extended details on the VAE work and it also demonstrates how VAE helps improve algorithms for solving practical problems.

1.1.2. Graphical Partitioning of the Natural Number Network

The natural number network is a fundamental structure in mathematics that provides a framework for understanding the properties of numbers and their relationships with one another. We discover a partitioning of the natural number network into disjoint clusters when visualizing natural numbers as rooted trees. A graphical natural number representation is

introduced based on a formal logic foundation of a recursively defined function that exposes a one-to-one mapping between natural numbers and rooted trees. According to the isomorphic concept of graph theory, a relation between pairs of natural numbers is then revealed and shown to partition the entire natural number network into disjoint finite tree sets (forests). This relation encapsulates in visual form of the natural structure and distribution of primes in a manner not previously investigated.

Theory attendant to these representations, as well as to efficient arithmetical operations for computing necessary intermediate values and a visual structure for storing them, is presented to increase the availability of these recursively defined representations. This content is based on our former published paper [52] with extended details on the VAE work and it also demonstrates the procedure how VAE helps discover new theories.

1.2. Software Environments

Our empirical studies are conducted through graphic programs created based on environments of Processing Foundation which aims to combine software literacy with visual arts. Software such as Processing and p5.js in this foundation are popular open-source and cross-platform sketchbooks in the creative coding area [34, 62]. Creative coding is a type of computer programming with the goal of creating something expressive instead of something functional. In this respect, Visualized Algorithm Engineering (VAE) is a kind of creative coding of algorithm itself which combines both aspects.

Both Processing and p5.js are used in the VAE works presented by this dissertation where Processing (a language based on Java) is used to create standalone graphic programs and p5.js (a JavaScript library) is used to create online graphic programs. Specifically, Processing is used to build a standalone graphic program for *Backbone Determination in Wireless Sensor Networks (WSNs)* problem which has several consecutive algorithm procedures. A Java-based standalone program allows us to pursue better performance such as processing millions of sensor nodes. For the *Graphical Partitioning of the Natural Number Network* problem, several individual visual tools and animations are created for computational analysis which

is suitable to use p5.js to build light-weight online programs.

1.3. Outline of The Thesis

The work is presented in four chapters. Chapter 2 focuses on the problem of Backbone Determination in Wireless Sensor Networks (WSNs). Chapter 3 focuses on the problem of Graphical Partitioning of the Natural Number Network. Chapter 4 discusses how visualization contributes in algorithm engineering. Each of the above chapters can be read separately and is independent of the other three chapters.

1.4. Contributions

The major contributions are listed in the following sections, with each section focusing on a specific category.

1.4.1. Backbone Determination in Wireless Sensor Networks (WSNs)

- (a) An efficient method is proposed for partitioning a Random Geometric Graph (RGG) into several disjoint bipartite subgraphs, each of which on average dominates $(1 - \epsilon)n$ nodes, based on a two-phase sequential coloring algorithm. All procedures of this method have a linear time $O(n)$ complexity. Additionally, this method addresses the *backbone determination in WSNs* problem by modeling a WSN as an RGG with the cluster-based routing protocol.

The two-phase sequential coloring algorithm has two procedures: smallest-last coloring and relay coloring. The relay coloring algorithm is a new algorithm invented based on the smallest-last coloring algorithm.

- (b) A side project from the *backbone determination in WSNs* problem has been engineered to create an RGG of a large amount of nodes (graphs with up to millions nodes have been tested). This algorithm engineering work is implemented using two methods (Sweep method and Cell method) which are originally referenced from Dhia Mahjoub's

thesis work [46].

Furthermore, the two algorithms are improved and extended to be applied in 2D and 3D geometries on multiple coordinate systems, including Cartesian, Polar, Cylindrical and Spherical. A customized Cell method algorithm has been invented to create an RGG specifically for the sphere geometry on the spherical coordinate system. The performances of the two algorithms have been analyzed for all mentioned coordinate systems (the Sweep method has $O(n^{\frac{3}{2}})$ time complexity and the Cell method has linear $O(n)$ time complexity.)

1.4.2. Graphical Partitioning of the Natural Number Network

- (a) A new tree structure termed “primordial tree” was discovered based on the automorphism of the rooted tree of Matula number.
- (b) The entire natural number network can be partitioned into mutually exclusive clusters of natural numbers. Each cluster of natural numbers are connected by a primordial tree with nodes each labeled one number of the set. The proof of this disjoint clustering is also provided.
- (c) The primordial tree structure also reveals an adjacency relation termed ”Integer Connectivity”: $i \cdot p_j \mathbb{R} j \cdot p_i$ (p_k is the k_{th} prime number), which applies to any pair of integers $i, j \in \mathbb{N}$. An algorithm for constructing a primordial tree based on the integer connectivity relation is also described.
- (d) A data structure termed “ $i \cdot p_j$ matrix” is provided to depict the relation between $i \cdot p_j$ ($i, j \in \mathbb{N}$), and it can be utilized to identify the number set that is clustered by a primordial tree.
- (e) A new number spiral termed “Primordial Spiral” is constructed, which is a graphical depiction of the set of centrum numbers (the number that labels the centrum node of a primordial tree).

1.4.3. Visualized Algorithm Engineering (VAE)

- (a) The terminology Visualized Algorithm Engineering (VAE) is proposed, which emphasizes the role of visualization in algorithm engineering. It is also a combination of creative coding and algorithm engineering.
- (b) The interactive features such as move, rotate and zoom in/out are designed and added to the animated algorithm visualizations. The VAE works on the two graph partitioning problems demonstrate how visualization helps to gain insights about the algorithm, tune heuristics for improving the performance, and finally solve the problem.
- (c) A complete Graphical User Interface (GUI) has been implemented for a standalone animation program in the VAE of the *Backbone Determination in WSNs* problem. This GUI has the potential to be utilized as a Processing language library that incorporates mouse, keyboard and touch controls. It also includes an auto-scaling feature to adapt to any high Dots Per Inch (DPI) displays.
- (d) Two visualizations, Primordial Constellation and Primordial Garden, have been created to offer a novel and unique perspective on the entirety of the natural number world.

Chapter 2

BACKBONE DETERMINATION IN WIRELESS SENSOR NETWORKS

2.1. Introduction

Wireless Sensor Networks (WSNs) are composed of low-cost, self-governing electronic sensors spread throughout a region where the sensors communicate with each other wirelessly. Sensors are typically thrown in an unattended and random manner across the area to be monitored. Wireless communication allows the formation of flexible networks, which can be deployed rapidly over wide or inaccessible areas. However, minimizing energy expenditure without compromising the quality of essential services such as area coverage and efficient routing remains to be a major design challenge that needs to be overcome for this technology to gain more traction in the real world. Nowadays, WSNs emerge as an active research area in which challenging topics involve energy consumption, routing algorithms, selection of sensor locations, and so forth [13].

In a WSN of a given topology (as Figure 2.1 shows), the sensor information is usually collected then forwarded to a base station known as a sink node. Collecting data from all sensors in a network necessitates constraints on the sensor distances, making efficient routing crucial, especially in cases where a significant number of sensors are involved. One popular solution is to perform routing only through a connected subset of nodes (called a “virtual backbone”) in the WSN [2,17,47]. Since only nodes of the virtual backbone (such as the blue and red nodes depicted in Figure 2.1) are responsible for packet forwarding and performing in-network services, other nodes can conserve energy by spending more time in a low-power idle mode. As a result, a virtual backbone can streamline the routing process, leading to decreased overall network energy consumption. Furthermore, it can serve other purposes such as transmission scheduling, broadcasting, and localization [71]. For a given WSN, we

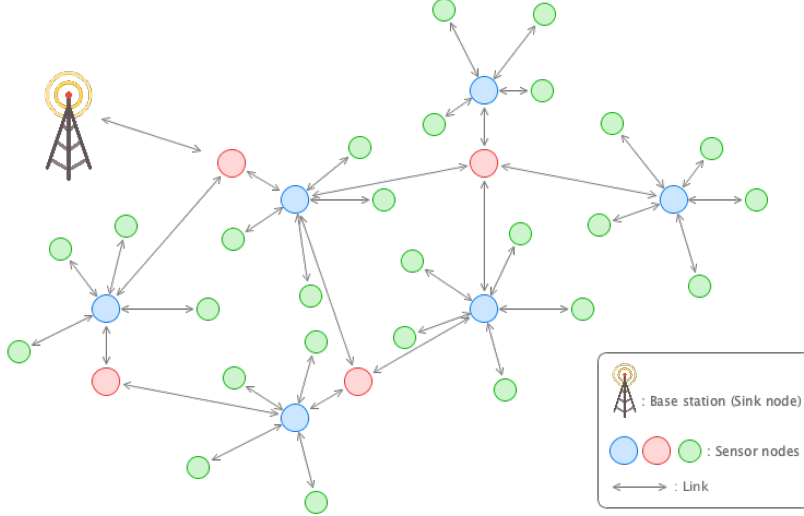


Figure 2.1: Wireless Sensor Network

typically wish to find a virtual backbone with a minimum number of nodes to maximize the potential energy saving [18]. However, successful routing requires that the nodes in the virtual backbone remain connected, and that every other node is connected to at least one backbone node. This arouses our interests in investigating the method of backbone determination which meets (or almost satisfies) the seemingly contradicting requirements.

Our contributions to this problem are achieved through Visualized Algorithm Engineering (VAE), which can be summarized in the following two aspects:

- We choose to use a unique planar bipartite structure (a bi-regular three and four lattice grid) to construct a virtual backbone. Each area of the region on which the WSN is deployed will be covered by at least 4 backbone nodes. The k -coverage ($k > 3$) increases robustness and also helps localization and synchronization [31, 43, 60].
- Instead of forming a single virtual backbone, we partition a WSN modeled as an Random Geometric Graph (RGG) into a limited number of densely packed disjoint backbones. Each backbone is a subgraph approximating the bipartite structure mentioned in the first bullet. The multi-partitioning also provides better scalability and versatility.

In Section 2.2, we will introduce the background of RGG and its cluster-based formation.

RGG is our preferred WSN model which is the computational foundation of our algorithms and visual programs. Cluster-based control structure provides an efficient way of routing, which is the basis of the virtual backbone.

In Section 2.3, the strategy for backbone formation will be proposed, which utilizes a unique bi-regular three and four lattice grid structure. This is a bipartite structure that provides nice topological k -coverage performance metrics. Thus, we will characterize the bipartite grid partition we seek to find in a large sparse RGG.

In Section 2.4, we will present the VAE of backbone determination with the following four procedures: sensor deployment, link determination, backbone partitioning and backbone refinement. A standalone 3D graphic program was created for the VAE work, which can be downloaded from <https://s2.smu.edu/~zizhenc/Project>. The source code is maintained on GitHub: <https://github.com/zizhenc/WSN>. Screenshots and videos are also provided.

In Section 2.5, we will conclude this work and analyze pros and cons of our backbone determination method. Furthermore, some directions for future works are also provided.

2.2. Background

2.2.1. Network Computational Model

Topologically, at any instant in time a network can be modeled as a graph $G(V, E)$ where V is a set of vertices representing nodes and E is a set of edges representing links between the nodes. Generally, the manner of sensor deployment can be either random or deterministic [39]. Although deterministic deployment seems to require fewer nodes to achieve a given degree of coverage and connectivity than the random deployment, it is impractical to devise a strategy whereby each sensor is deployed precisely at some location. Besides, random deployment has also been shown to have a low transfer delay [58]. Therefore, random graphs are preferred in modeling Wireless Sensor Networks (WSNs). Several researchers have contributed studies of such models [7, 32, 48, 57, 61]. The Erdő-Rényi model is one such graph $G(n, p)$ where each possible edge among n nodes has probability p of existing [75].

Geographically, two nodes at closer distance have a higher probability of being connected, so Random Geometric Graph (RGG) seems more realistic than the Erdö-Rényi model. RGG denoted $G(n, r)$ chooses random n locations of nodes uniformly and independently in a region and two nodes are connected by an edge if they are within a Euclidian distance less than a certain threshold r [57]. In the context of a WSN, every node in an RGG is equipped with an omnidirectional antenna that monitors a small disc area with a radius of r . This disc area encompasses other nodes that are connected to the node through edges in the graph, which represent links. The radius is assumed to be sufficiently small due to the constraints on the distance of sensors, which yields a large sparse RGG.

Realistically, the RGG model is still oversimplified due to the constant r value. The radio signal in a WSN is disrupted by either large scale fading effects such as reflection, shadowing, diffraction and scattering or small scale fading effects such as interference, heterogeneous powers and transmission errors [69]. Therefore, each node might have inhomogeneous transmission range (i.e., r should not be consistent). Having inhomogeneous transmission ranges causes the corresponding graph to be directed which complicates routing. One solution published in the proceedings of the ACM Mobihoc 2003 [9] is to consider the connectivity induced only by the symmetric edges (each edge denotes the minimum transmission distance between the two connected sensors) to avoid directed graph models. Taking this into account, the results based on the RGG model are still applicable in practice.

2.2.2. Cluster-based Formation

Cluster-based control structure allows a more efficient use of resources especially when a network consists of a large amount of sensors [13]. Clustering consists of partitioning a network into a number of clusters achieved by grouping nodes inside a certain transmission area (in the case of an RGG, it is a disc with radius r). A cluster is a locally connected group of nodes with a leading node (known as “cluster-head”) from which each of the other nodes (known as “slave nodes”) is one hop away. Thus, data collected by slave nodes can be directly sent to their cluster-heads. Slave nodes can shut down some functionalities and

hence save power while their cluster-heads are providing a service (e.g., in-network data processing and synchronization) [69]. Therefore, topologically, we expect more slave nodes to save more energy in a WSN.

Routing performed via a virtual backbone consisting of cluster-heads is also called “cluster-based routing” [67, 72]. Generally, there are two ways of cluster-based routing. One is to route via cluster-heads directly [72], that is, the set of cluster-heads has power to be connected and hence forms a virtual backbone itself. The other way is to select another group of nodes as routing relays (or as gateways in some WSN strategies [56]) to the cluster-heads. We call such node set a “relay set” and the set of cluster-heads is called a “primary set” accordingly. The union of a primary set and a relay set is a connected set that forms a virtual backbone. Although the first way has a simpler structure, it requires the selected cluster-heads to preserve higher energy or more resources than other nodes to fulfill both in-network processing and routing. A longer distance of transmission might also be necessary, which causes more interference. Therefore, we choose the second way and design algorithms based on the corresponding definition of a virtual backbone.

Definition 2.1 *A **virtual backbone** of a WSN of cluster-based control structure is a connected subset of nodes that are composed of a set of cluster-heads (primary set) and a set of routing relays (relay set). Such a backbone network provides a path for the exchange of information between sensor nodes and sink nodes.*

In Figure 2.1, each blue node is a cluster-head that controls its adjacent green nodes as slave nodes, and each red node is a routing relay to its adjacent blue nodes. The union of blue and red nodes forms a virtual backbone that routes the network’s information to the base station (sink node).

2.3. Backbone Formation

Given that the sensor network is modeled as a Random Geometric Graph (RGG), we can use strong concepts like independence and domination to construct a virtual backbone. In the case of cluster-based routing, a virtual backbone is composed of a set of cluster-heads

(i.e., primary set) and a set of routing relays (i.e., relay set). In graph theory, a Dominating Set (DS) is a set of nodes in a graph such that any node of the graph is either an element of the DS or has a neighbor in the DS. Hence the set of cluster-heads constitutes a DS, and a virtual backbone by definition is a Connected Dominating Set (CDS). To minimize energy expenditure, the CDS with smallest cardinality, called Minimum Connected Dominating Set (MCDS), is the optimal configuration of a virtual backbone [18]. Unfortunately, finding an MCDS in a given connected graph is known to be NP-hard [30]. Several researchers have contributed algorithms of constructing a virtual backbone by approximating an MCDS in a Wireless Sensor Network (WSN) [16, 70, 74]. Therefore, our strategy of backbone formation is to select a paired primary and relay set to approximate an MCDS with desired goals of network longevity and continuous multi-coverage.

2.3.1. Primary and Relay Bipartite Set

In graph theory, an independent set is a set of nodes in a graph such that no two nodes of the set are adjacent. A Maximal Independent Set (MIS) must also be a Minimal Dominating Set (MDS) that can be used to constitute a set of cluster-heads with small cardinality. As clusters are not necessarily to be disjoint, an MDS also assures minimal overlap between clusters. Nodes belonging to multiple clusters can be selected as routing relays between the respective cluster-heads. Routing relays should be separated far enough (not connected) from each other in order not to cause more interference. Therefore, both the primary set and the relay set are ideally modeled as independent sets. The union of a primary set and its paired relay set is a connected bipartite set that forms a virtual backbone. Deployment with the bipartite feature has the property that allows two frequency channels to route with no co-channel interference in the backbone.

Generally, the deployment (geometry) of a virtual backbone in a WSN can be various according to different application requirements. In our case, k -coverage of the virtual backbone is one of mainly considered performance metrics that is topology related. In literature, a network is considered to have k -coverage when each point within the region is covered by

a minimum of k sensors [43, 58]. In our specific use cases, we are evaluating the k -coverage of a virtual backbone for a WSN, which should be defined as every node in the WSN being covered by at least k nodes from the virtual backbone. Requiring k -coverage will increase accuracy of tracking, improve robustness or fault-tolerance, and provide better localization and synchronization [31, 43, 60]. Figure 2.2 provides two regular bipartite planar grids (a Cartesian lattice and a Hexagon lattice) where the blue nodes form a primary independent set and the red nodes form a relay independent set. The Cartesian and Hexagon lattice grids

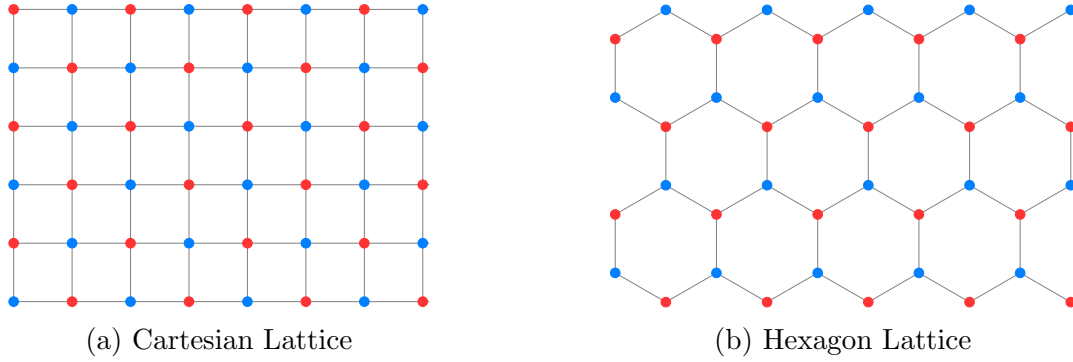


Figure 2.2: Manually Placed Bipartite Lattice Grids

are 4-coverage and 6-coverage deployments respectively.

Employing idealized results from models of cellular systems, the triangular lattice grid provides a maximum density of nodes [45]. Figure 2.3(a) shows a closest-packed primary set of an RGG $G(n, r)$ where each node is at distance r' that is slightly greater than the radius r . Accordingly, Figure 2.3(b) shows a relay set where each node is at distance r''

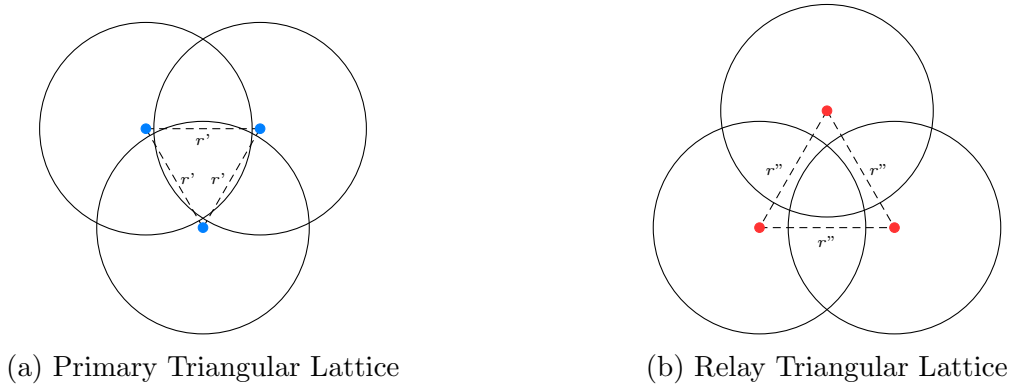


Figure 2.3: Triangular Lattice Independent Sets

that is slightly greater than r' . Figure 2.4 illustrates that this primary set deployment is a 3-coverage set and the relay set deployment is a 1-coverage set. Thus, combining the primary

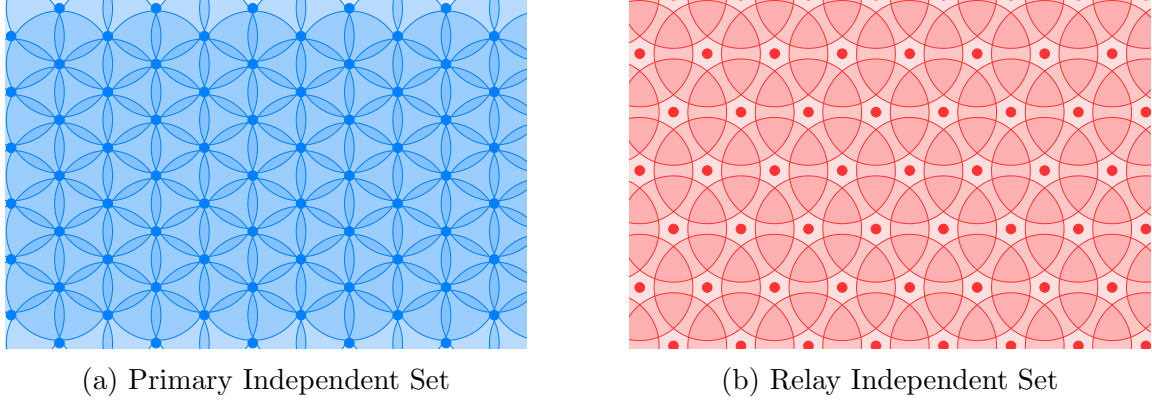


Figure 2.4: Coverages of Primary Set and Relay Set

and relay set, an alternative deployment termed “bi-regular three and four lattice grid” (since it has bi-regular degree three and four) shown in Figure 2.5 forms a virtual backbone that has 4-coverage. Any node of the WSN will be covered by k nodes of this backbone, where $4 \leq k \leq 7$. If only the nodes of the primary set provide in-network services, then the virtual

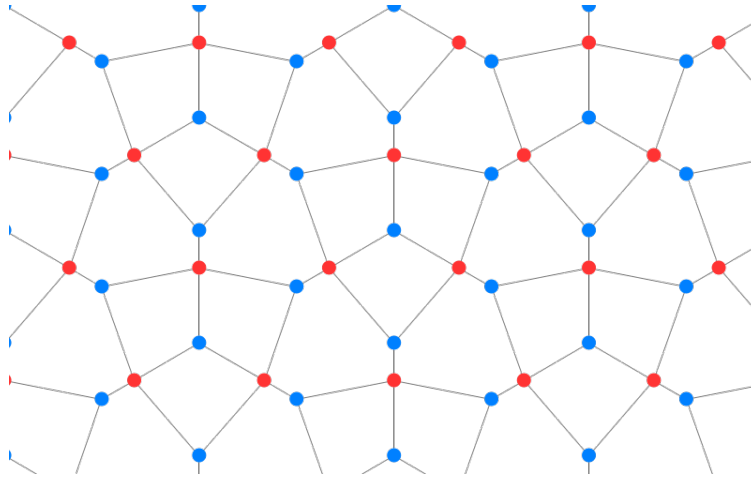


Figure 2.5: Bi-regular Three and Four Lattice Grid

backbone will still have 3-coverage ($3 \leq k \leq 4$).

2.3.2. Backbone Determination via Multi-partitioning

The deployment of bi-regular three and four lattice grids provides a highly effective

backbone that satisfies the multi-coverage requirements exceptionally well. However, it's usually impractical to perform routing in a WSN via a manually placed backbone grid. Moreover, a single backbone will quickly exhaust its energy reserves while serving a whole network. Ideally, it is desirable that the grid can be offset and replicated k times to form k backbones using all nodes of the WSN. Efficient algorithms are necessary to provide a more robust scheme that devises a sleep/activity schedule by selecting a maximum collection of disjoint backbone grids and sequentially rotates their activity (duty cycle) to ensure full coverage and enlarge the whole network lifetime [76]. Therefore, our goal is to partition a WSN modeled as an RGG into several disjoint virtual backbones each approximates an MCDS (i.e., bi-regular three and four lattice grid), which can be mathematically stated as below.

We model a WSN by an RGG $G(n, r)$ with a node set V formed by choosing n locations over a region in a uniform random manner, and an edge set E formed by introducing an edge between every node pair whose Euclidian distance is less than r . Let $\{V_1, V_2, \dots, V_k\}$ be a partition of V into disjoint sets. Each set V_i ($1 \leq i \leq k-1$) induces a connected bipartite subgraph of the RGG composed of a pair of primary and relay independent sets. Specifically, we shall term $\{V_1, V_2, \dots, V_{k-1}\}$ a Bipartite Component Partition (BCP) denoted $BCP(\delta, \epsilon)$ of the $G(n, r)$ if the union of V_i ($1 \leq i \leq k-1$) comprises $(1 - \delta)n$ nodes of V and if the induced subgraphs $\langle V_i \rangle$ ($1 \leq i \leq k-1$) each on average dominates $(1 - \epsilon)n$ nodes of V . V_k is the residual node set not employed by the BCP. Our goal is to determine such a partition $BCP(\delta, \epsilon)$ with suitably small δ and ϵ .

Our method of backbone partitioning (i.e., BCP) utilizes graph coloring algorithms to produce color classes each representing an independent set of nodes. Certain coloring algorithms are shown to generate a collection of similar-sized disjoint and verifiably dominating sets that have equally good performance in covering the network with minimal overlap [47]; hence these sets constitute good candidates for constructing duty-cycled virtual backbones. To satisfy the requirements outlined in Definition 2.1, we adopt a two-phase sequential coloring algorithm to partition an RGG into separate backbone grids. In the first phase, we

utilize the smallest-last coloring algorithm [51] to select primary sets, while in the second phase, we use a novel sequential coloring algorithm called the “relay coloring” algorithm to choose paired relay sets. Previous research has demonstrated that the smallest-last coloring algorithm is superior to other sequential coloring algorithms in generating partitions of independent sets that form nearly triangular lattices [47, 48]. Relay coloring algorithm is an adaptive sequential coloring algorithm designed to provide relay sets paired with the primary sets resulting from the smallest-last coloring algorithm. Details of engineering are given in Section 2.4.3.

2.4. Visualized Algorithm Engineering

We characterize the backbone determination problem by specifying a Random Geometric Graph (RGG) and a desired Bipartite Component Partition (BCP) whose existence is demonstrated by an algorithm engineering work that visualizes graph models and partitioning algorithms. A graphic program is created through Processing language, which not only presents animated algorithmic procedures, but also allows us to move, rotate and zoom in/out the 3D network models. Limited by the paper print media, we can only show screenshots and links of pre-recorded videos here. The standalone program is archived at <http://s2.smu.edu/~zizhenc/Project>. Our approach to backbone determination is divided into four main procedures: (1) sensor deployment, (2) link determination, (3) backbone partitioning, and (4) backbone refinement. Each following subsection presents one procedure with algorithmic analysis and visualized results.

2.4.1. Sensor Deployment

Sensor deployment is modeled by placing nodes over a region, which determines two properties of a network: geometry and sensor locations. We consider Wireless Sensor Networks (WSNs) on four surfaces (shown in Figure 2.6):

- (a) Unit Square. A square has side length $l = 1$. Unit square is a fundamental geometry that helps to describe, analyze and visualize algorithms. It’s a 2D planar surface that

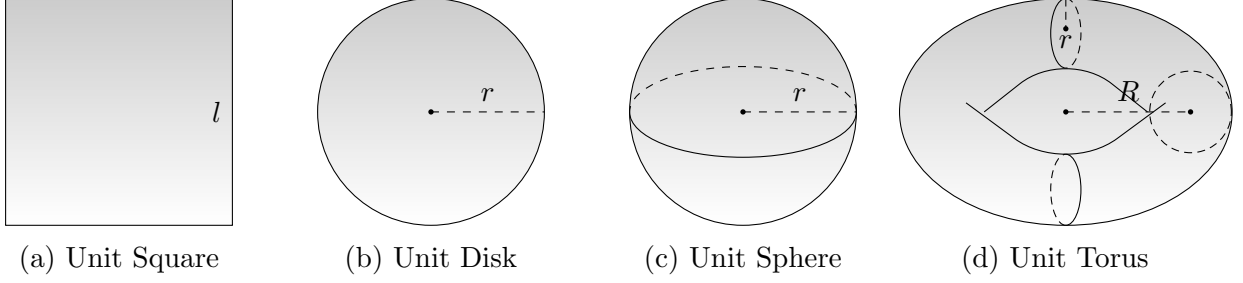


Figure 2.6: Models of Four Geometries

allows simulating WSNs on flat regions. However, there are small degree biases caused by its four corners and boundary (also called “border effects”) [35].

- (b) Unit Disk. A disk has radius $R = 1$. It has almost the same features as the unit square except that it removes the four corner small degree biases.
- (c) Unit Sphere. A sphere has radius $R = 1$. Unlike the unit square or disk, the unit sphere is a 3D geometry that has no biases or border effects. In graph theory, it allows an easy count of faces through the Euler characteristic χ [26]. In applications, the unit sphere can also model a global sensor network for planetary explorations [19, 59].
- (d) Unit Torus. We term a torus which has major radius $R_M = 1$ and minor radius $R_m = 1/2$ as “uint torus”. It is a unique 3D geometry with numerous applications [1]. Similar to the unit sphere, it allows face counting directly by the Euler characteristic χ and is commonly used to model WSNs to avoid border effects [55, 77].

Modeling a WSN as an RGG requires a uniform random distribution of nodes over the surfaces, which could be tricky for different geometries. In the case of Cartesian coordinate system, we need the x, y parametric definition of a geometry to calculate locations of nodes. For a unit square which is parametrically defined by

$$\begin{cases} x(u, v) = u \\ y(u, v) = v \end{cases} \quad u, v \in \left[-\frac{1}{2}, \frac{1}{2}\right]$$

, we can use two uniform random number generators to compute u and v to generate uniformly distributed nodes. However, in the case of a unit disk, it is incorrect to use the parametric definition

$$\begin{cases} x(\rho, \theta) = \rho \cdot \cos\theta \\ y(\rho, \theta) = \rho \cdot \sin\theta \end{cases} \quad \rho \in [0, 1], \theta \in [0, 2\pi)$$

to generate uniform random nodes. The area element is given by $dA = 2\pi\rho d\rho$, which gives a concentration of nodes in the center (as Figure 2.7(a) shows). Therefore, the correct

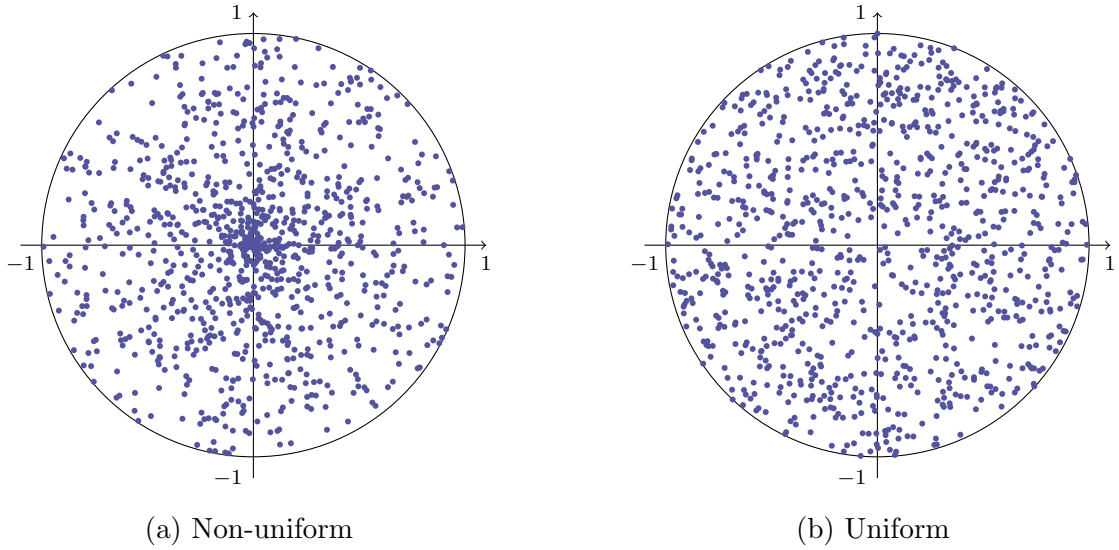


Figure 2.7: Random Nodes in Unit Disk

definition should be

$$\begin{cases} x(\rho, \theta) = \sqrt{\rho} \cdot \cos\theta \\ y(\rho, \theta) = \sqrt{\rho} \cdot \sin\theta \end{cases} \quad \rho \in [0, 1], \theta \in [0, 2\pi)$$

where the area element is $dA = \pi d\rho$ (as Figure 2.7(b) shows). The parametric definitions of all four surfaces for generating uniformly distributed nodes are listed in Table 2.1 [41, 49].

The uniform random distribution of an RGG $G(n, r)$ also helps us to determine the radius r . For any planar surface, we prefer to specify the expected average degree \overline{deg} for

Table 2.1: Definitions or Formulas of Geometries

Geometry	Parametric Definition (Cartesian System)	Surface Area	r
Unit Square	$\begin{cases} x(u, v) = u \\ y(u, v) = v \end{cases} \quad u, v \in [-\frac{1}{2}, \frac{1}{2}]$	1	$\sqrt{\frac{\overline{deg}+1}{n\pi}}$
Unit Disk	$\begin{cases} x(\rho, \theta) = \sqrt{\rho} \cdot \cos\theta \\ y(\rho, \theta) = \sqrt{\rho} \cdot \sin\theta \end{cases} \quad \rho \in [0, 1], \theta \in [0, 2\pi)$	π	$\sqrt{\frac{\overline{deg}+1}{n}}$
Unit Sphere	$\begin{cases} x(u, \theta) = \sqrt{1-u^2} \cdot \cos\theta \\ y(u, \theta) = \sqrt{1-u^2} \cdot \sin\theta \\ z(u, \theta) = u \end{cases} \quad \theta \in [0, 2\pi), u \in [-1, 1]$	4π	$2\sqrt{\frac{\overline{deg}+1}{n}}$
Unit Torus	$\begin{cases} x(\theta, \phi)(1 + \frac{\cos\theta}{2}) \cdot \cos\phi \\ y(\theta, \phi)(1 + \frac{\cos\theta}{2}) \cdot \sin\phi \\ z(\theta, \phi) = \frac{\sin\theta}{2} \end{cases} \quad \theta, \phi \in [0, 2\pi)$	$2\pi^2$	$\sqrt{\frac{2\pi(\overline{deg}+1)}{n}}$

the given n nodes to determine an RGG, which provides a convenient density parity over alternative surfaces for comparison of the number and quality of the virtual backbone grids in the BCP. Because of the uniform random distribution, the relation between any area and the contained node amount can be described by Equation (2.1) where n_1 and n_2 represent the node amounts within the two areas S_1 and S_2 , respectively.

$$\frac{S_1}{S_2} = \frac{n_1}{n_2} \quad (2.1)$$

$$\frac{\pi r^2}{S} = \frac{\overline{deg} + 1}{n} \Rightarrow r = \sqrt{S \cdot \frac{\overline{deg} + 1}{n\pi}} \quad (2.2)$$

Therefore, for any node u in the middle area of a surface (no border effects), let the surface area be S and the sensor transmission area be S_u for the node u , Equation (2.2) can be derived according to the illustration of Figure 2.8. When considering border effects, as illustrated by Figure 2.9, we have the following observation.

Observation 2.1 *It can be observed that the degree of boundary nodes on average is typically around half of the average degree (\overline{deg}) of an RGG, whereas corner nodes are subject to greater border effects and thus exhibit a degree of approximately one-fourth of \overline{deg} .*

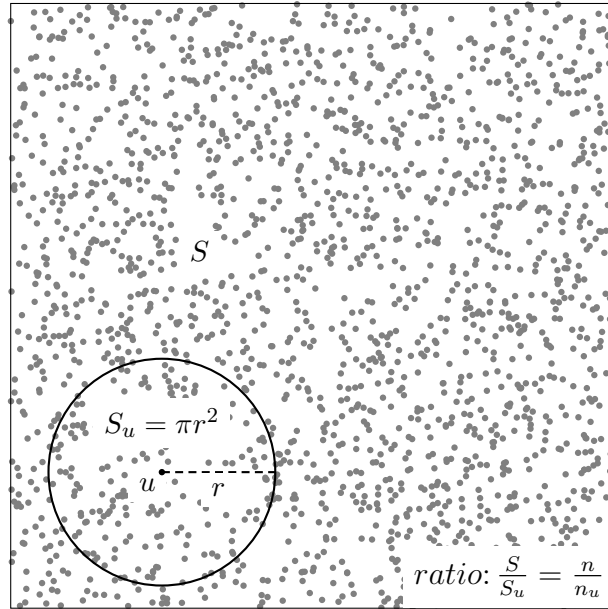
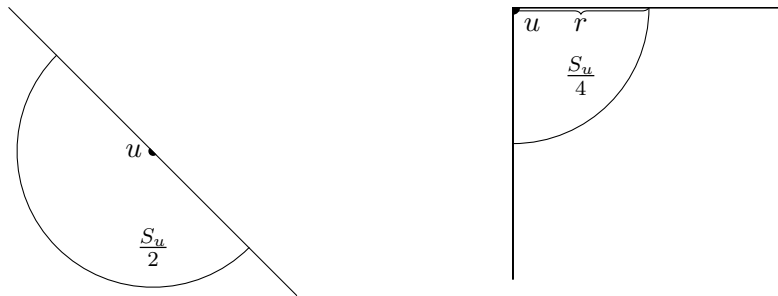


Figure 2.8: Ratio of Areas and Nodes



(a) Boundary Node

(b) Corner Node

Figure 2.9: Degree Estimation of Nodes in Border Area

Table 2.1 lists the formulas of surface areas and corresponding r values of all four geometries. Figure 2.10 shows screenshots of sample RGGs of 6400 nodes on four surfaces with r values computed to yield expected average degree 100 in each case. Video 2.1 demonstrates the

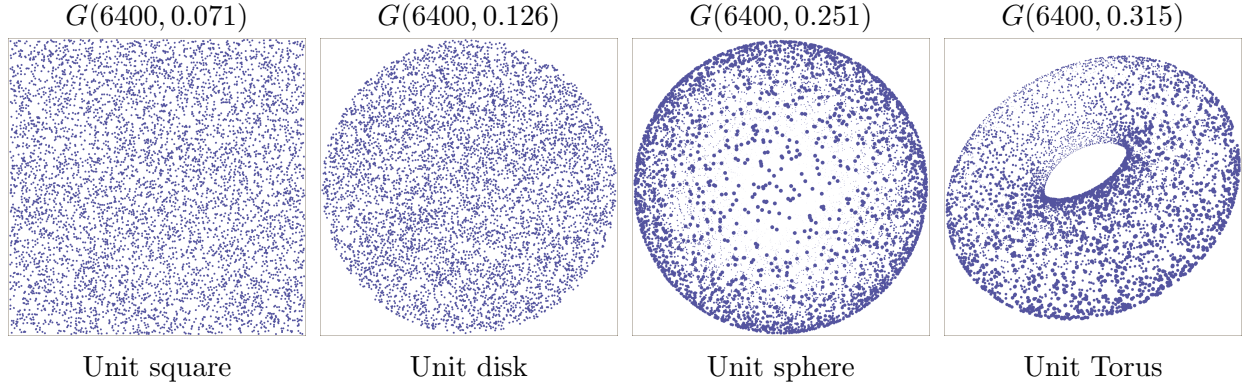
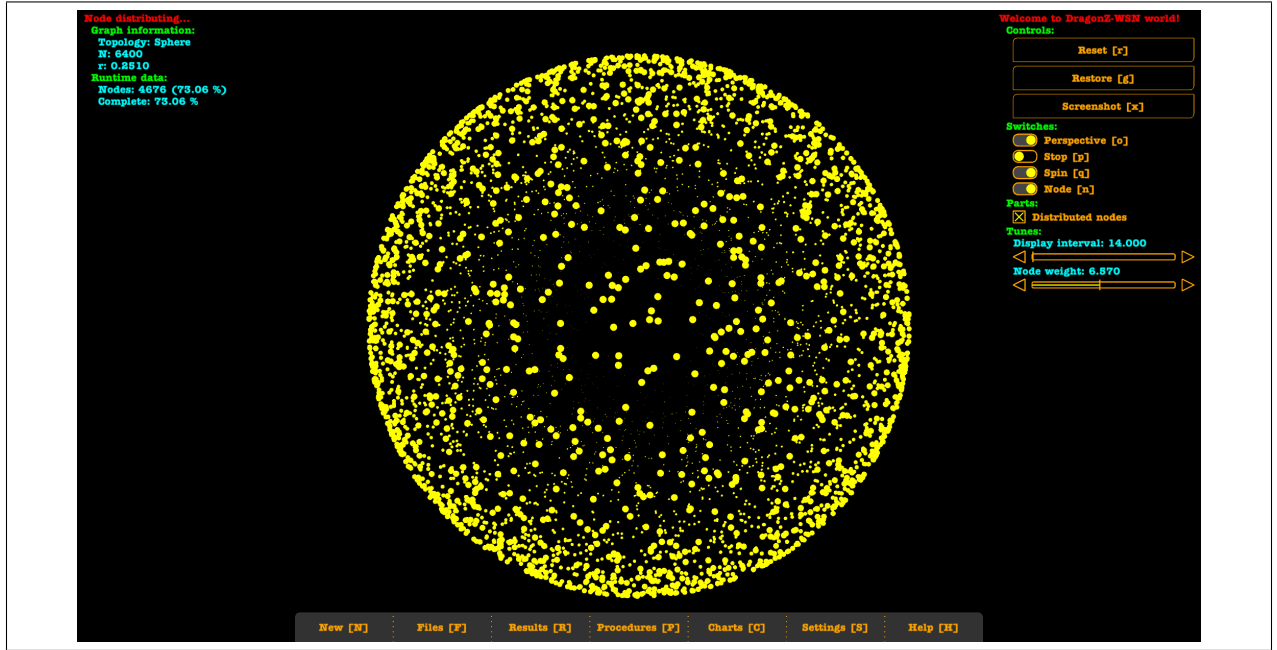


Figure 2.10: Random Node Distribution on Four Surfaces

sensor deployment animation on the four surfaces listed in Figure 2.10.



Video 2.1: Sensor Deployment

2.4.2. Link Determination

Once sensors are deployed, we need to determine links to generate the network. As Section 2.2.1 stated, an RGG $G(n, r)$ that is used to model a WSN is assumed to be a

large sparse graph by having sufficiently small radius r . Therefore, we choose to use the adjacency list data structure to store the RGG to save space. A trivial brute force algorithm of establishing the adjacency information (i.e., determining links) is that every node u checks every other node v of the remaining $(n - 1)$ nodes and creates an edge (u, v) if the distance $d(u, v) \leq r$, which has time complexity of $O(n^2)$. Although the brute force approach is within polynomial time, it's still not efficient if n is fairly large. We implemented the brute force approach in our program as a verification reference to the other two more efficient approaches “sweep method” and “cell method” respectively.

2.4.2.1. Sweep Method

In a 2D Cartesian coordinate system, the sweep method orders the nodes v_1, v_2, \dots by their x coordinates and then determines the neighbors of each node v_i in this order by checking the nodes v_{i+1}, v_{i+2}, \dots whose x coordinates have distance from v_i at most r . As Figure 2.11(a) illustrates, a vertical strip formed by the nodes to be checked is moving along the x -axis while running the sweep algorithm. To consider the time complexity, let's assume

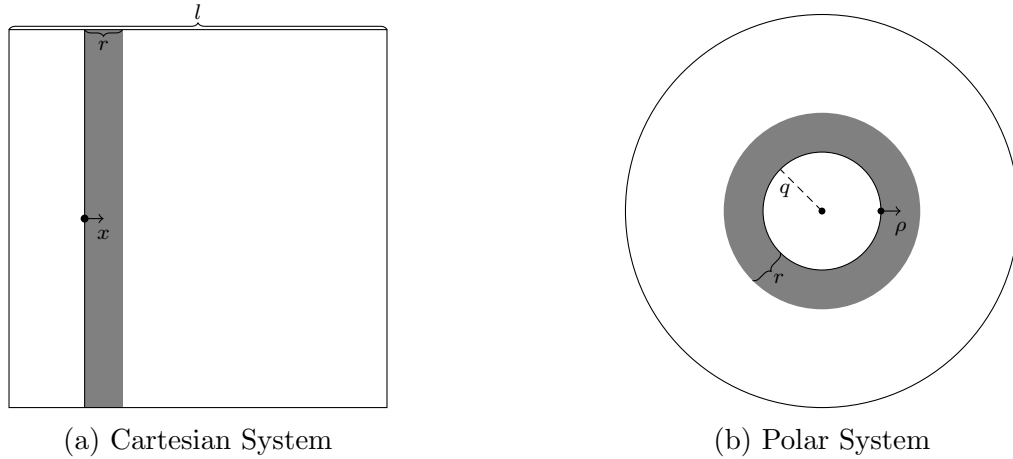


Figure 2.11: Sweep Method on 2D Coordinate Systems

the majority nodes of the RGG are in its inscribed rectangle. According to Equation (2.1), if the rectangle has length l , then the expected number of nodes in the strip is $\frac{nr}{l}$, which provides a time complexity of $O(n^2r)$. Ignoring any border effects, nr^2 is a constant value (deduced from Equation (2.2)), then the sweep algorithm has time complexity of $O(n^{\frac{3}{2}})$.

The sweep method offers two advantages. Firstly, it works consistently in the same manner for all higher dimensions (including those beyond three). Secondly, it can be used with other coordinate systems as well. For example, in the polar coordinate system, each point (ρ, ϕ) is determined by a distance ρ from the pole and an angle ϕ from the polar axis. Then we can order the nodes by their ρ coordinates and determine the neighbors of each node v in this order by checking the ring of nodes whose ρ coordinates have distance from v at most r (illustrated in Figure 2.11(b)). To consider the time complexity, let's assume the majority nodes of the RGG are in its inscribed disk. If the disk has radius R and the inner circle of the ring has radius q , then the expected number of nodes in the ring checked by each node will be $\frac{(2q+r)nr}{R^2}$, which provides a time complexity of $O(n^{\frac{3}{2}})$ (ignoring any border effects). Cylindrical coordinate system and spherical coordinate system will be supported in a similar way (as Figure 2.12 shows) with the same time complexity*. Tables 2.2

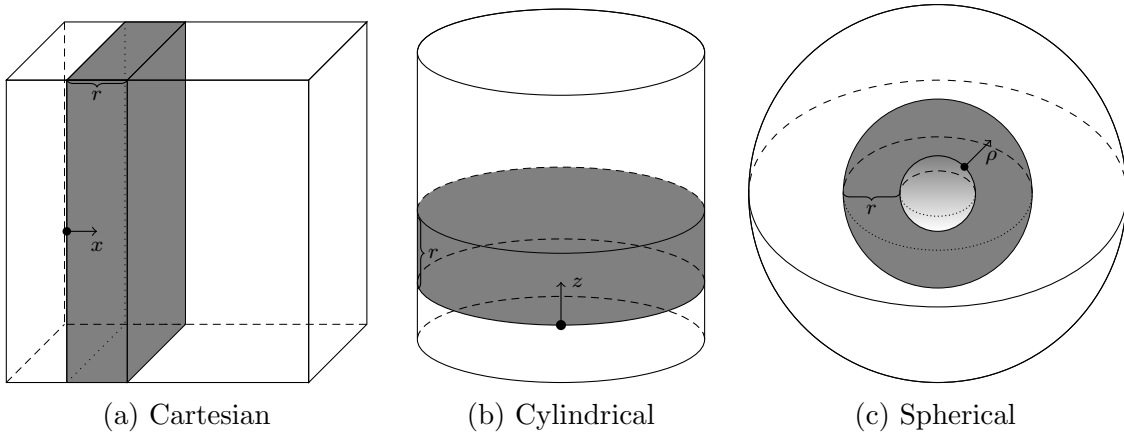


Figure 2.12: Sweep Method on 3D Coordinate Systems

to 2.5 present screenshots of sweep algorithmic procedures applied on four distinct surfaces in multiple coordinate systems.

*Here the time complexity is calculated under the condition of having a majority of nodes in the inscribed regular geometry of the coordinate system (rectangle for a 2D Cartesian system, disk for a polar system, cuboid for a 3D Cartesian system, cylinder for a cylindrical system and ball for a spherical system). Otherwise, the sweep method might actually be the brute force method (consider the nodes uniformly distributed on a straight line along the y -axis in a Cartesian coordinate system).

†The sweep method applied to a unit sphere in a spherical coordinate system is actually the brute force method.

Table 2.2: Unit Square Sweep Method


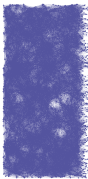
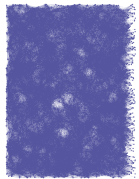
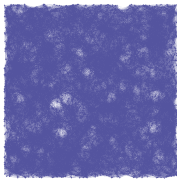
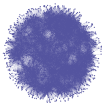
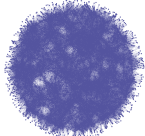
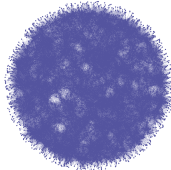
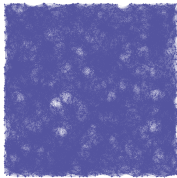
$G(6400, 0.071)$	25%	50%	75%	100%
Cartesian System				
Polar System				

Table 2.3: Unit Disk Sweep Method



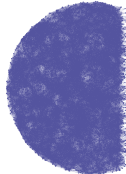
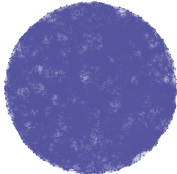

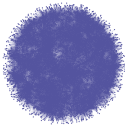
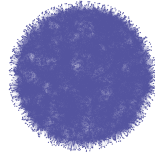
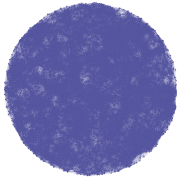
$G(6400, 0.126)$	25%	50%	75%	100%
Cartesian System				
Polar System				

Table 2.4: Unit Sphere Sweep Method[†]

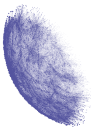
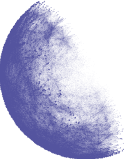
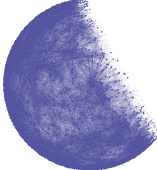
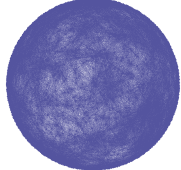
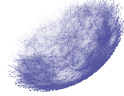
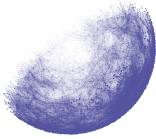
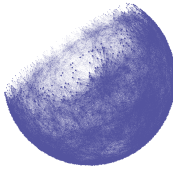
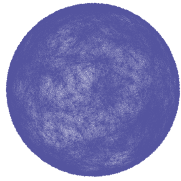
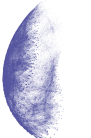

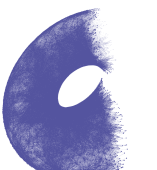
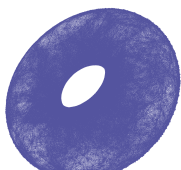
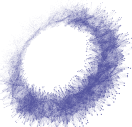
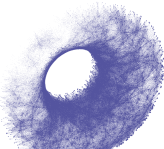
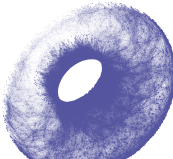
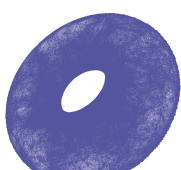


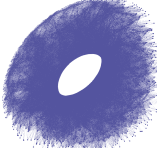
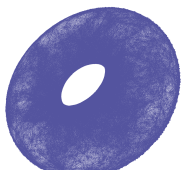
$G(6400, 0.251)$	25%	50%	75%	100%
Cartesian System				
Cylindrical System				

Table 2.5: Unit Torus Sweep Method

$G(6400, 0.315)$	25%	50%	75%	100%
Cartesian System				
Cylindrical System				
Spherical System				

2.4.2.2. Cell Method

In a 2D Cartesian coordinate system, the cell method divides the circumscribed rectangle of the RGG shape into r by r square cells and each node is put in the appropriate cell as in bucket sort. As Figure 2.13(a) illustrates, the neighbors of each node are determined by checking the nodes of all cells the circle touches (the red cell and its surrounding eight cells). According to Equation (2.1), each cell will contain $\frac{nr^2}{S}$ nodes at most where S is the

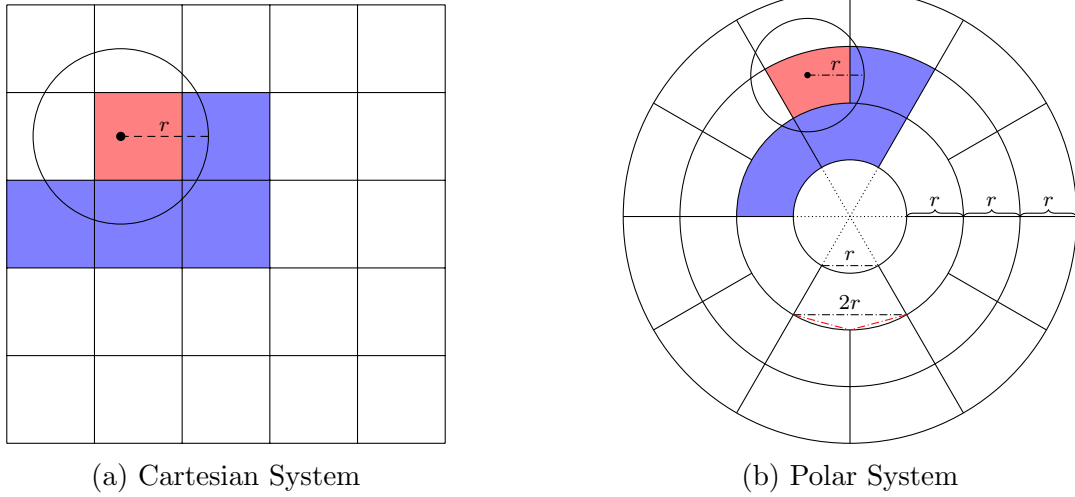


Figure 2.13: Cell Method on 2D Coordinate Systems

area covered by the RGG. This gives us a time complexity of $O(n^2r^2)$. Ignoring any border effects, nr^2 is a constant value (deduced from Equation (2.2)), then the cell method is a linear time ($O(n)$) algorithm.

The cell method is more efficient than the sweep method as it computes only a small expected number of node-pair distances per edge determination. Besides, with a sequential operation, each node in the red cell only needs to check the nodes in the blue and red cells to determine its neighbors (as Figure 2.13(a) shows). Technically, cell method is also a multi-threaded algorithm since the nodes of each cell (or each group of cells) can determine their neighbors in a separate thread. However, it might require a customized cell division strategy to achieve its expected linear time efficiency for each different geometry and each different coordinate system.

Figure 2.13(b) shows a cell division strategy for the polar coordinate system. First, divide

the circumscribed disk of an RGG shape into multiple annuli each has width r (the center is a circle of radius r). Second, evenly divide the disk into six fans, which in turn gives us six even sectors per annulus. Each sector of the innermost annulus has an inner arc of width r . This provides us initial cells of the disk (one central circle and the remaining annulus sectors). It's obvious that the further out an annulus is, the larger each of its sectors is. Once an annulus sector cell is large enough (when the width of its inner arc equals $2r$), we then evenly split it into two sector cells where each has an inner arc of width slightly larger than r (as the two red dash dotted segments illustrate). Similarly, with a sequential operation, each node in the red cell only needs to check the nodes in the blue and red cells to determine its neighbors.

Figure 2.14 shows cell division strategies for 3D coordinate systems where (a) and (b) are directly extended from the strategies of 2D coordinate systems, which can be used to determine edges of RGGs of any 3D geometries. However, only 3D solid geometries (we

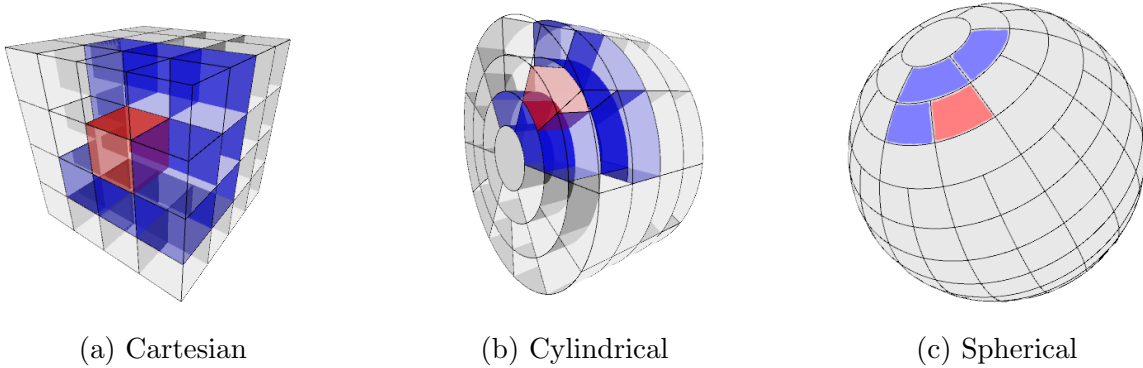


Figure 2.14: Cell Method on 3D Coordinate Systems

implemented a unit cube and a unit ball in the program for verification purpose) are optimized for the two strategies. Either strategy (a) or (b) may waste a lot of space if applied to our 3D surface geometries (i.e., the unit sphere and the unit torus) since most cells contain few or none nodes (consider the inner volume of a unit sphere). Inspired by the division strategy of the polar coordinate system, we use the same way to divide cells on the half surface of a sphere in a spherical coordinate system (as Figure 2.14(c) shows), then the cell method can sequentially run on the northern and southern hemispheres simultaneously from

the poles to the equator (as Figure 2.15 shows). This cell division strategy of a spherical

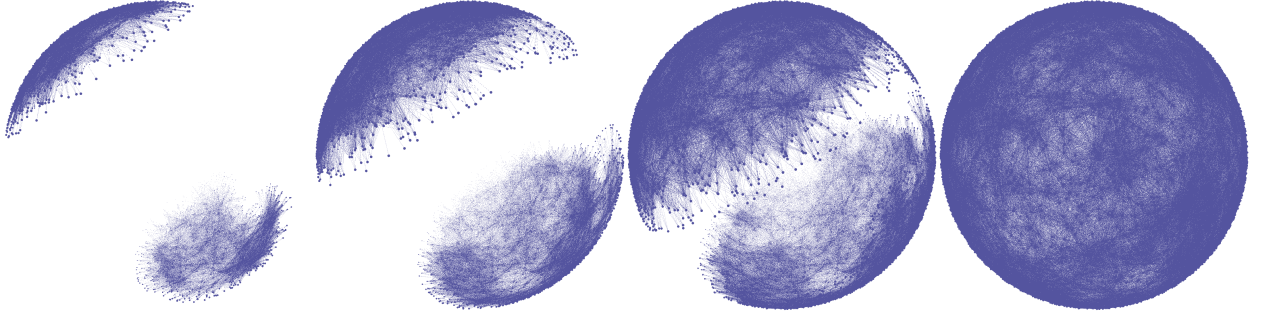


Figure 2.15: Unit Sphere $G(6400, 0.251)$ Cell Method

coordinate system is customized only for any sized sphere geometry. Tables 2.6 to 2.9 present the screenshots of cell algorithmic procedures applied on four distinct surfaces in multiple coordinate systems. Video 2.2 showcases the link determination animation on these four

Table 2.6: Unit Square Cell Method

$G(6400, 0.071)$	25%	50%	75%	100%
Cartesian System				
Polar System				

distinct surfaces, where each animation compares the sweep method and cell method across multiple coordinate systems.

2.4.3. Backbone Partitioning

We introduce an efficient (linear time) algorithm including a two-phase sequential coloring procedure (smallest-last coloring and relay coloring) that employs only the topology (not the geometry) of an RGG to provide a desired BCP. A sequential coloring algorithm in graph

Table 2.7: Unit Disk Cell Method



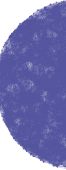
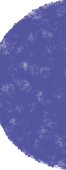




$G(6400, 0.126)$	25%	50%	75%	100%
Cartesian System				
Polar System				

Table 2.8: Unit Sphere Cell Method

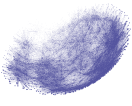
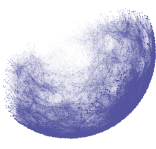
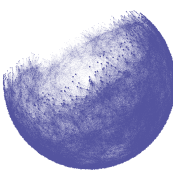
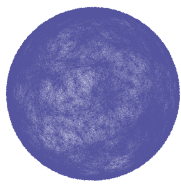
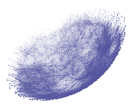
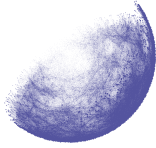
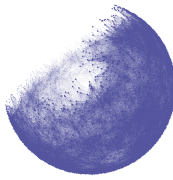
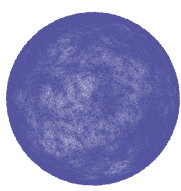
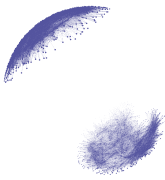
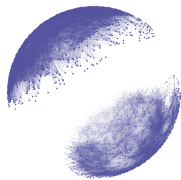
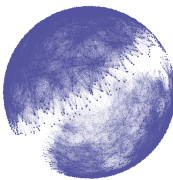
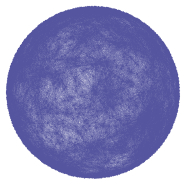
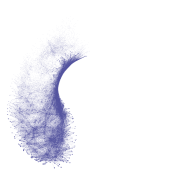
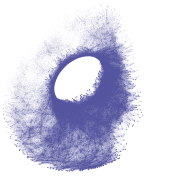
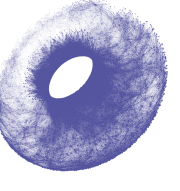
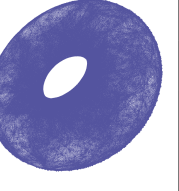
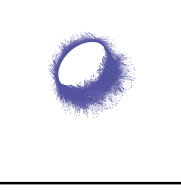
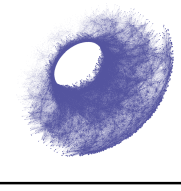
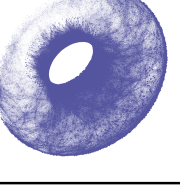
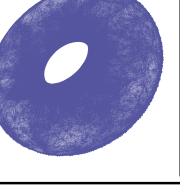
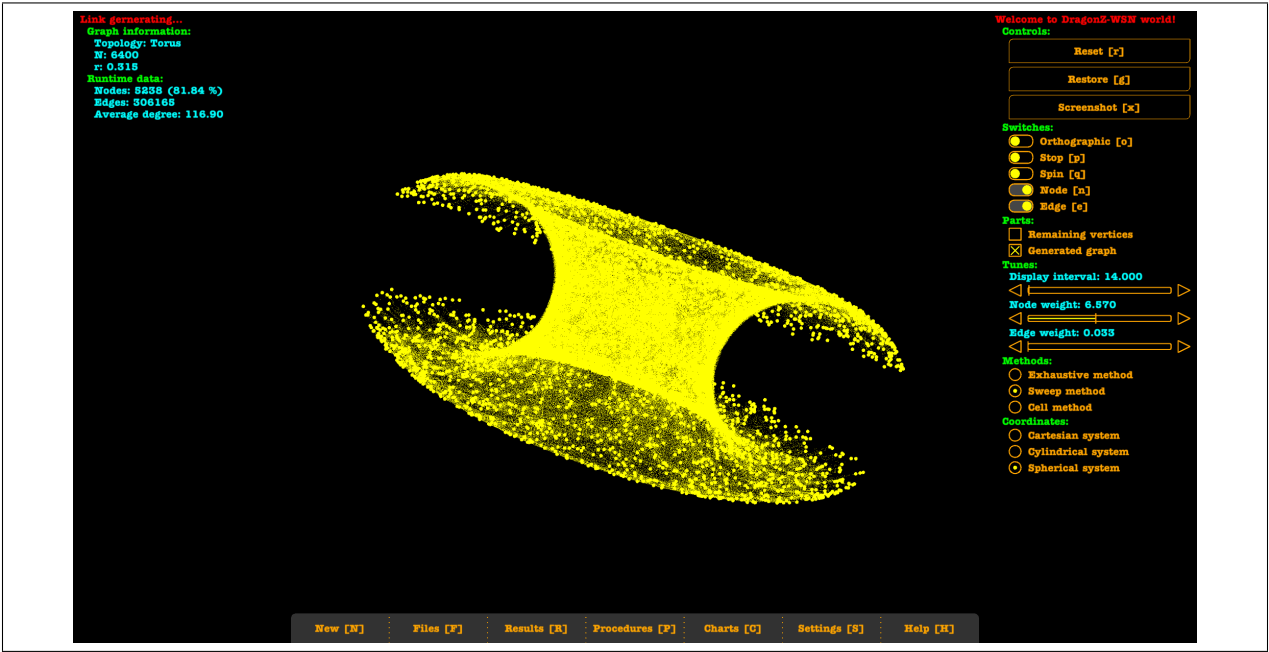
$G(6400, 0.251)$	25%	50%	75%	100%
Cartesian System				
Cylindrical System				
Spherical System				

Table 2.9: Unit Torus Cell Method

$G(6400, 0.315)$	25%	50%	75%	100%
Cartesian System				
Cylindrical System				



Video 2.2: [Link Determination](#)

theory is an algorithm that operates in the following two stages:

- Determine a sequence of the nodes of the graph;
- Color the graph according to the sequence in a greedy manner.

The process of coloring a graph involves the assignment of colors to its nodes in a way that ensures adjacent nodes are assigned distinct color values. Here the greedy manner is characterized by assigning the smallest color value to a given node that hasn't been assigned to its neighbors. The smallest-last coloring algorithm provides candidates of primary independent sets, while the relay coloring algorithm provides the paired relay independent sets for the selected primary sets. In the following Section 2.4.4, we will use an RGG $G(10000, 0.057)$ of unit square surface as a sample to describe the VAE in details. Tables and videos of related results on all surfaces will also be presented.

2.4.3.1. Smallest-last Coloring

The "smallest-last coloring" algorithm employs a greedy approach to color a graph. It utilizes a sequence generated by the "smallest-last ordering" algorithm, which sequentially removes one node with the minimum degree from the remaining graph and then places it at the beginning of the sequence. This removing process continues until the graph is empty, as depicted in Figure 2.16. Algorithm 1 is a pseudocode that contains two while loops where the

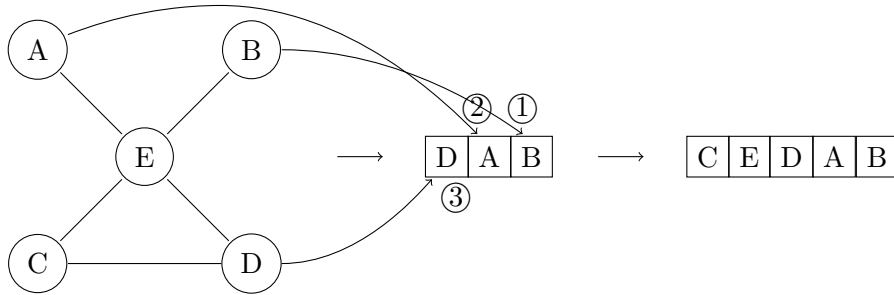


Figure 2.16: Smallest-last Ordering

first loop is the smallest-last ordering procedure and the second loop is the greedy-coloring procedure.

Algorithm 1 Smallest-last Coloring

```
1: function SLCOLOR(Graph  $G$ )
2:    $S \leftarrow$  new List;
3:    $T \leftarrow$  new Stack;
4:    $D \leftarrow$  new Degree-List( $G$ );
5:   while  $D \neq \emptyset$  do
6:     Remove a smallest degree node  $u$  from  $D$ ;
7:     Update all neighbors of  $u$  in  $D$ ;
8:      $T.push(u)$ ;
9:   while  $T \neq \emptyset$  do
10:     $u \leftarrow T.pop()$ ;
11:    Greedy-color  $u$ ;
12:     $S \leftarrow S + u$ ;
13:  return  $S$ ;
```

A data structure that maps degree values to lists of nodes (illustrated by Figure 2.17) termed “degree list” is introduced to achieve linear time efficiency of the smallest-last ordering algorithm. The degree list can be created on the fly as the adjacency information of an

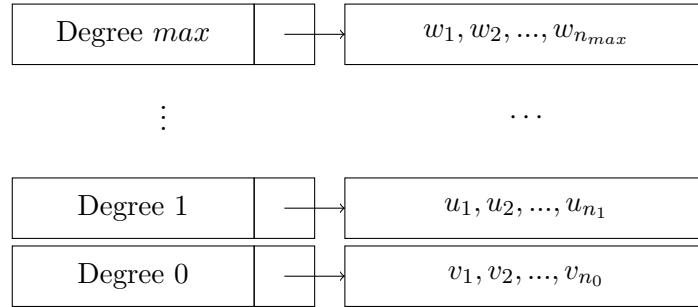


Figure 2.17: Degree List

RGG is being created, which helps to retrieve a minimum-degree node of the graph instantly (in $O(1)$ time). Once a minimum-degree node is being removed, each node of its neighbors’ needs to update the position in the degree list (jumps from the current degree list to the next lower degree list). There are two approaches to achieve a linear time implementation of the updating process. One option is to construct the degree list through a hash map data structure. It’s worth noting, however, that hash maps may experience reduced performance in the presence of hash value collisions. The other option is to create a degree list by using an array of doubly linked lists, where the index of the array indicates the degree value of

each list. Each element (i.e., node) in a doubly linked list contains pointers (or references in languages like Java) to its previous and next nodes, making it possible to remove a given node from its list and append the node to another list in constant time ($O(1)$). We implemented the second approach in our program, and Figure 2.18 illustrates the smallest-last ordering procedure of the sample RGG. This suggests that the ordering procedure typically

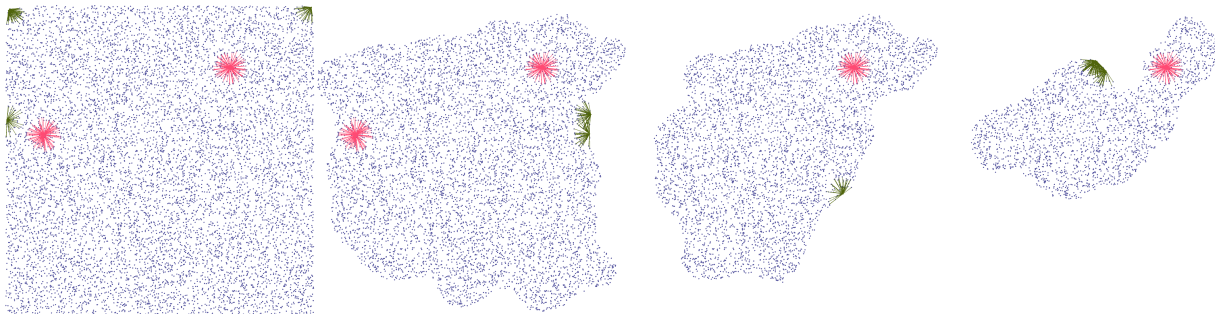


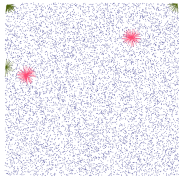
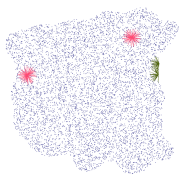
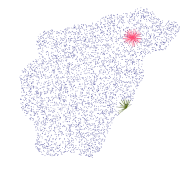

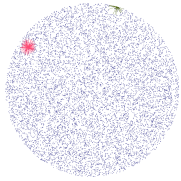
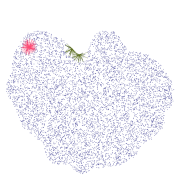

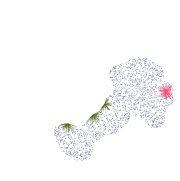
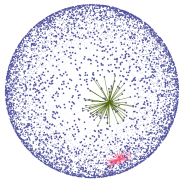
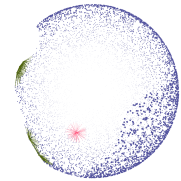
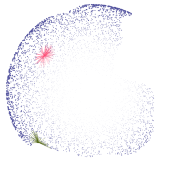
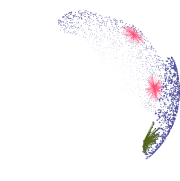
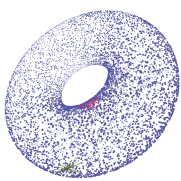
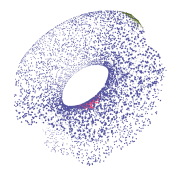
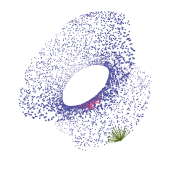
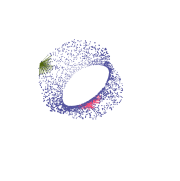
Figure 2.18: Smallest-last Ordering[‡]

starts at the surface boundary, gradually reducing its size until it disappears. The rationale behind this is evident — nodes on the boundary are more likely to have smaller degrees. Table 2.10 presents screenshots of the smallest-last ordering on four distinct surfaces.

Figure 2.19 displays several node-degree plots of the ordering procedure. The blue plot illustrates the original degree of each node, while the green plot displays the average original degree of all preceding nodes. The red plot represents the average degree of the remaining graph when removing each node, and the purple plot shows the degree of each node upon removal. The plots present that a sequence of nodes with a fairly stable number of neighbors will be provided by the smallest-last ordering algorithm, which yields a large complete subgraph (termed “terminal clique”) generally close or equal to the chromatic number [51]. Comparing the red and purple plots, the degree of each removed node is approximately half of the average degree of the remaining graph. This observation aligns with the fact illustrated by Figure 2.18, which shows that the graph shrinks from its boundary. Observation 2.1 further supports this finding, indicating that the degree of boundary nodes is about half of the average degree. The degree of each node upon removal during this procedure

[‡]The edges of the RGG are not displayed here for cleaner presentation. Nodes with the maximum and minimum degree are highlighted using red and green links, respectively.

Table 2.10: Smallest-last Ordering

Geometry	25%	50%	75%	100%
Unit Square $G(10^4, 0.057)$				
Unit Disk $G(10^4, 0.101)$				
Unit Sphere $G(10^4, 0.201)$				
Unit Torus $G(10^4, 0.252)$				

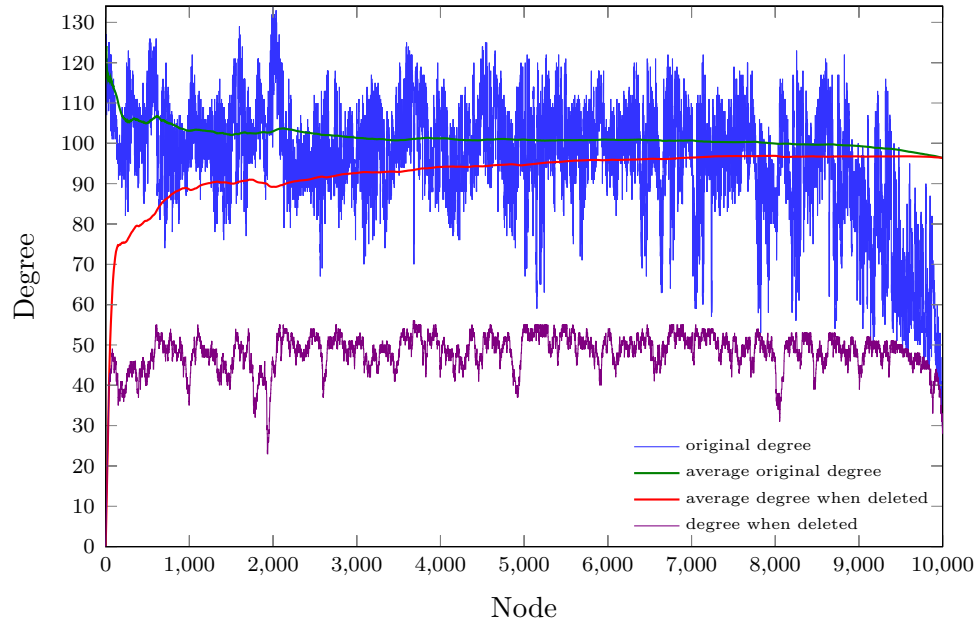
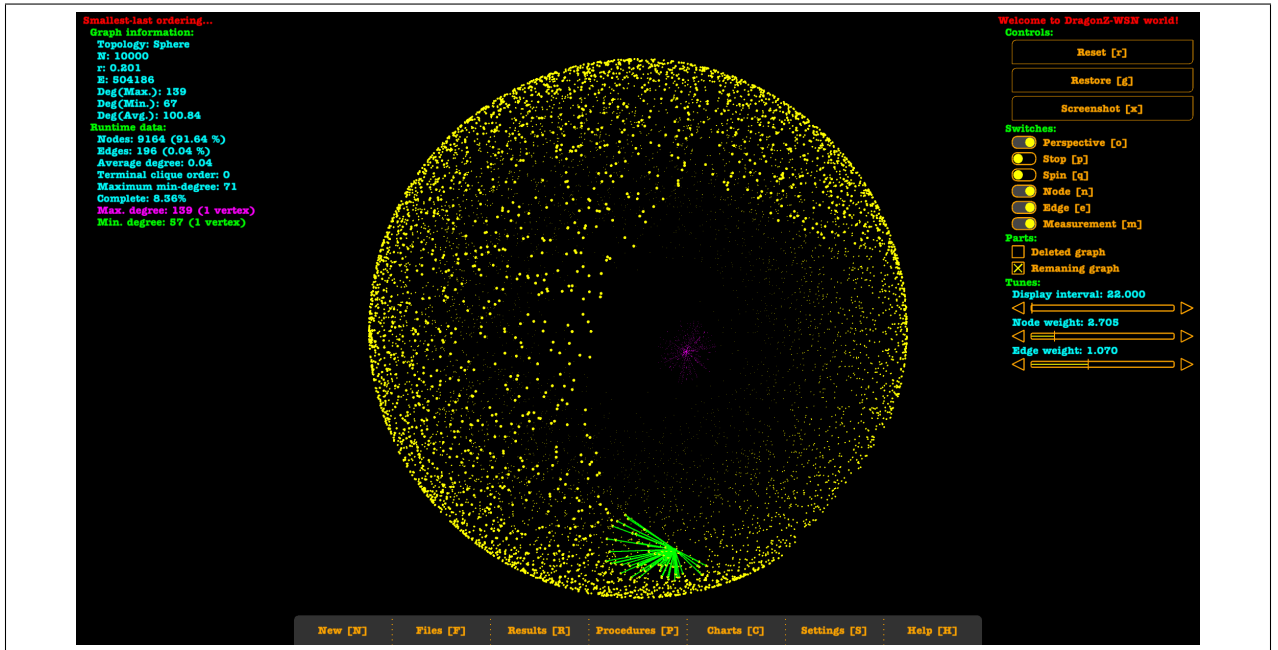


Figure 2.19: Node-degree Plots of Smallest-last Ordering

is less than or equal to the maximum value k of the purple plot, which computes a k -core or k -degenerate graph [5, 44]. Therefore, we would love to term the smallest-last ordering procedure as “Graph Degeneracy” procedure, as it reflects the “removing” feature of the process. The graph degeneracy procedure will always end into a terminal clique where the order (number of nodes) of the clique is indicated by the final (most left) heap value of the purple plot. Video 2.3 demonstrates the animation of this degeneracy procedure on the four different surfaces listed in Table 2.10, which also shows the node-degree plots for all the surfaces.



Video 2.3: RGG Degeneracy

After the nodes have been ordered, the coloring procedure starts from the resulting sequence v_1, v_2, \dots, v_n (provided by the stack in Algorithm 1), to color v_i in a greedy manner. When assigning a possible smallest color to v_i by traversing its neighbors, only the neighbor nodes preceding v_i in the smallest-last order (i.e., any neighbor v_k where $k < i$) need to be checked. This means, only the remaining neighbors of v_i require checking to assign a possible smallest color value to v_i in the degeneracy process. This corresponds to the “degree when deleted” plot, which is the purple plot shown in Figure 2.19. In the most extreme scenario, when all neighbors of the node “when deleted” with degree k are assigned distinct colors,

the number of colors used would be $k + 1$. Hence, the maximum value k of the purple plot indicates the possible maximum number of colors that could be used in the smallest-last coloring algorithm. Conceptually, the k -degenerate graph is equivalent to the coloring number k of the graph [24, 44]. Obviously the time complexity is $O(|V| + |E|)$ where V is the node set and E is the edge set of the graph. Figure 2.20 shows the coloring procedure of our sample RGG and it is structurally a reverse procedure compared to Figure 2.18. Table 2.11



Figure 2.20: Smallest-last Coloring[†]



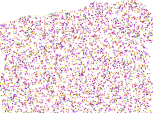







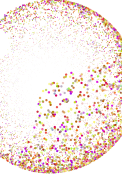
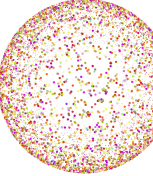



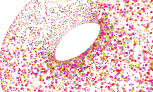
presents screenshots to compare the smallest-last coloring applied on four distinct surfaces.

Smallest-last coloring compares favorably to several other sequential methods (i.e., largest-first ordering, lexicographic ordering and random ordering) in providing compact packed independent sets of similar size and structure for over half amount of the total nodes [47, 53]. Figure 2.21 shows a color-size plot that presents the number of nodes of each color set resulting from the greedy coloring of our sample RGG. Each of the initial several color sets (Color #0 \sim #18) has roughly the same amount (around 261) of nodes and the number of nodes in these sets occupies about half amount of the nodes (49.66%) of the RGG. The size of each remaining color sets (i.e., Color #19 \sim #51) starts decreasing. Furthermore, we find out that most nodes in each initial color set are close-packed in a locally triangular lattice and that satisfies our desired characteristics for primary sets. To prove this, Definition 2.2 is introduced for us to quantify triangular lattice.

Definition 2.2 *For a planar graph drawn without edges crossing, the **degree of a face** is the number of edges bordering the face.*

We can apply Gabriel rules to each color set, which constructs Gabriel graphs. In graph

Table 2.11: Smallest-last Coloring Procedure

Geometry	25%	50%	75%	100%
Unit Square $G(10^4, 0.057)$				
Unit Disk $G(10^4, 0.101)$				
Unit Sphere $G(10^4, 0.201)$				
Unit Torus $G(10^4, 0.252)$				

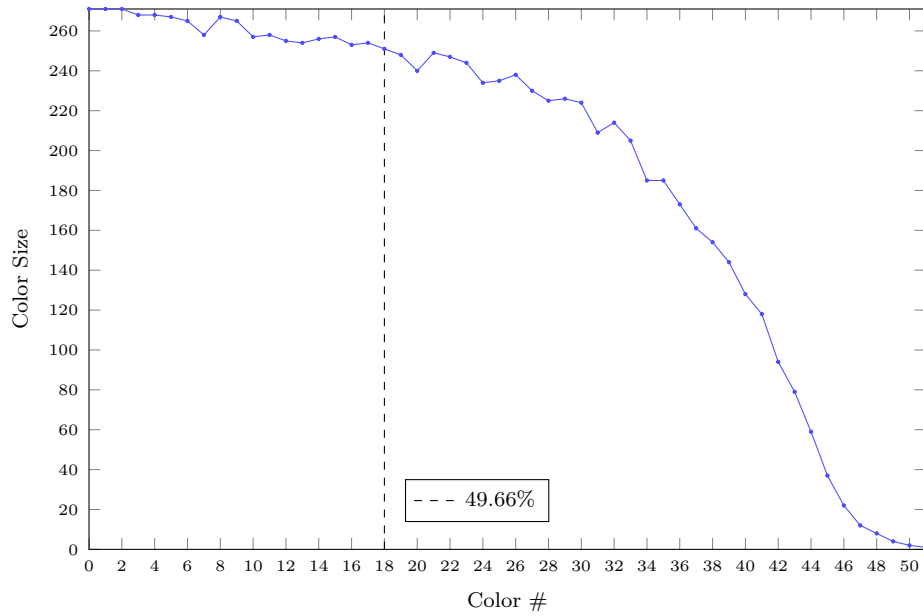


Figure 2.21: Color Size Plot of Smallest-last Coloring

theory, Gabriel graph is a spanning subgraph of the Delaunay triangulation [54], so if most nodes are packed in triangular lattice, then most faces of the Gabriel graph should have degree three. Figure 2.22 shows the Gabriel graphs with the percentage of degree-three faces $|f(3)|$ generated from the evenly selected four sample initial color sets resulting from our sample RGG. Video 2.4 demonstrates the smallest-last coloring animation with resulting

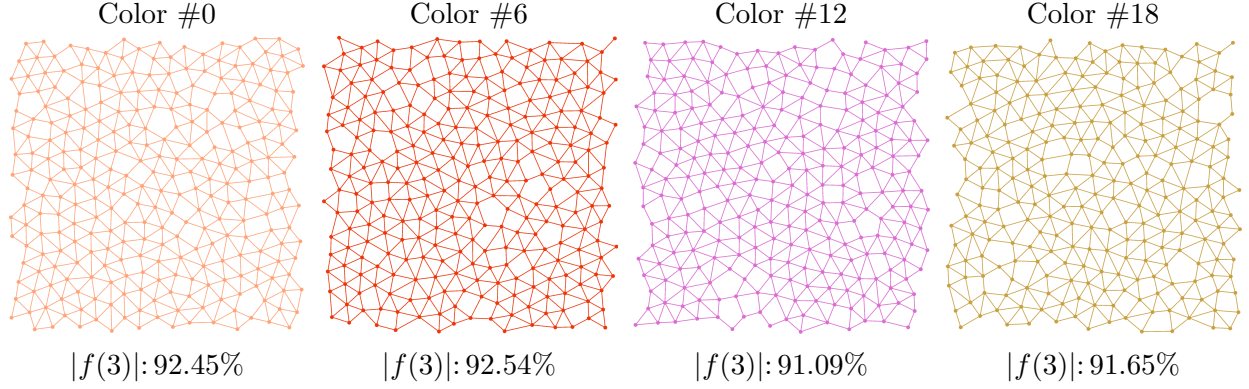


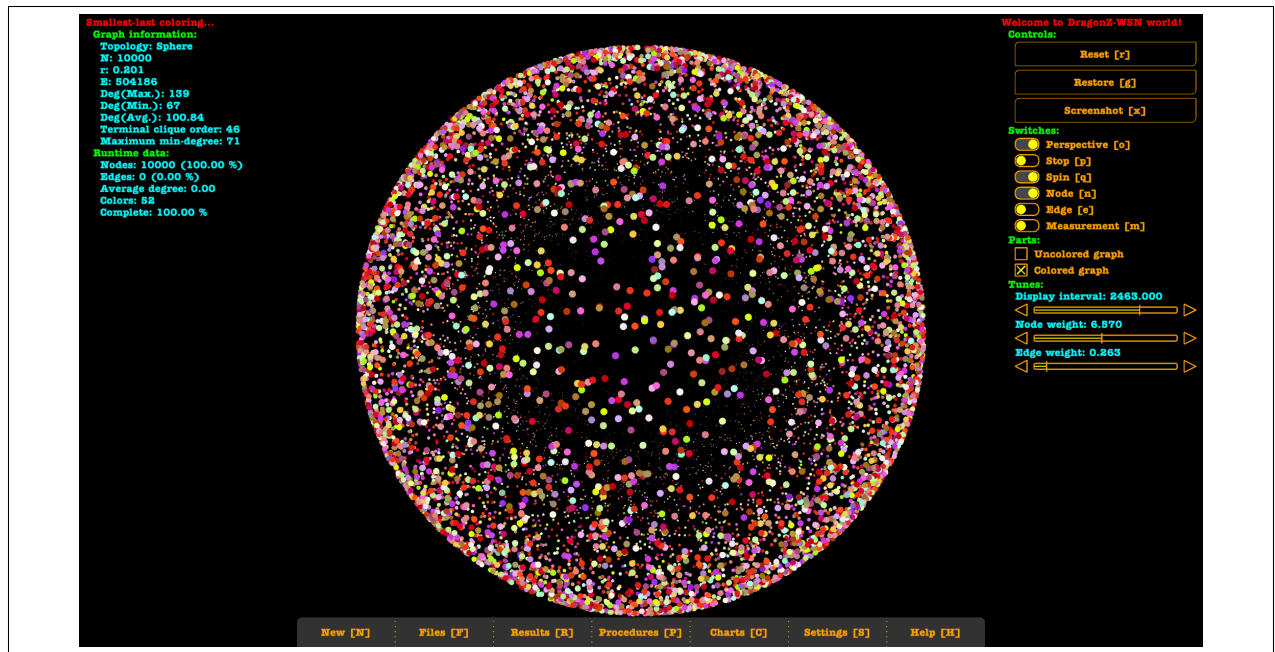
Figure 2.22: Degree-3 Faces of Selected Primary Color Sets with Gabriel Rules

color sets and color-size plots on the four different surfaces listed in Table 2.11. The initial two color sets of each surface will be applied Gabriel rules to demonstrate the density of triangular faces in this animation.

With these observations, we have the following definition of the primary independent sets.

Definition 2.3 *The **primary sets** of a WSN modeled as an RGG are the initial k color sets resulting from the smallest-last coloring procedure that covers around half amount of total nodes in the network.*

If we randomly select the remaining color sets (e.g., Color #19 \sim #51 in Figure 2.21) as paired relay independent sets, then the results are unpredictable. Figure 2.23 shows the bipartite subgraphs of four sample remaining color sets paired with Color #0 set where (a) and (b) have few disconnected components but (c) and (d) are distorted in pieces. Besides, the number of degree-three faces of each remaining color set will decrease as illustrated by Figure 2.24, since the smaller the color size is, the looser-packed nodes the color set contains.



Video 2.4: [Smallest-last Coloring](#)

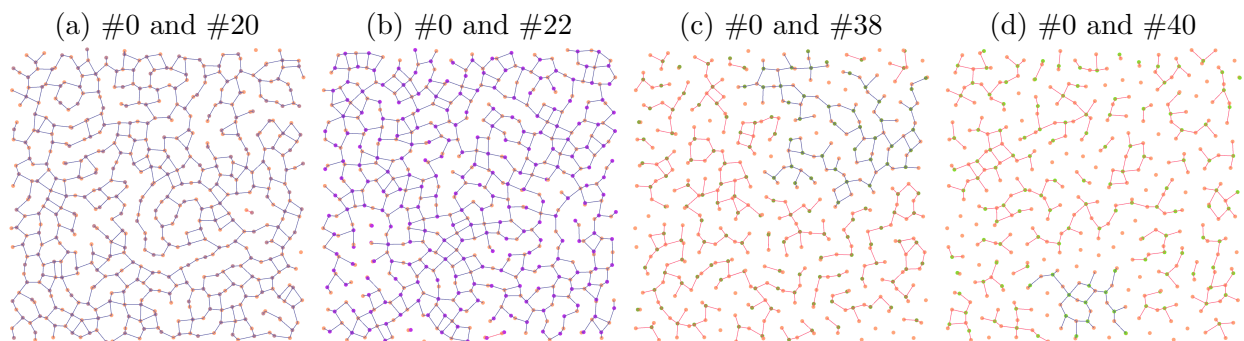


Figure 2.23: Random-paired Selected Bipartite Subgraphs

Therefore, the last several color sets are not close-packed in triangular lattice, which does

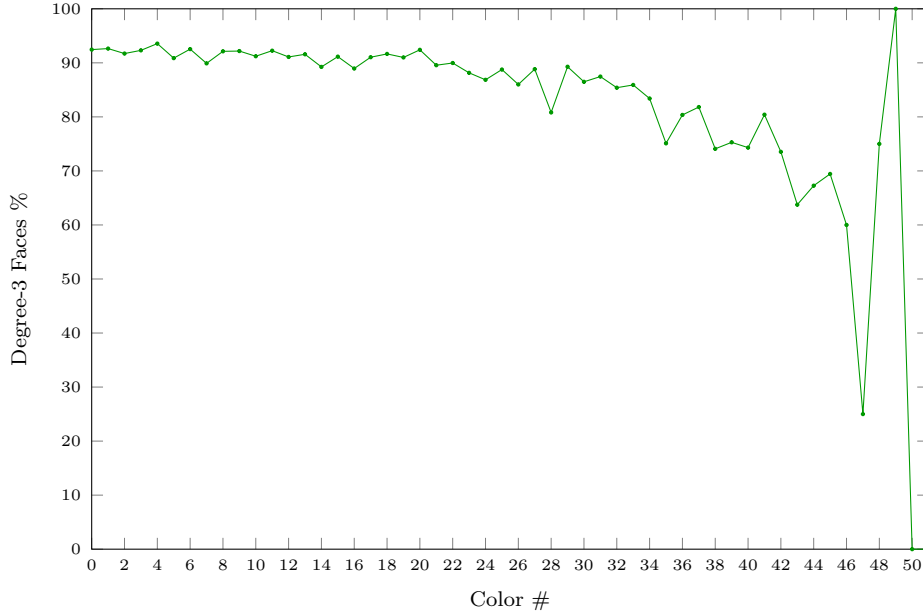


Figure 2.24: Degree-3 Faces Percentage Plot[§]

not meet our expectation. This motivates us to reorganize the nodes of the remaining color sets (termed “relay candidates”).

2.4.3.2. Relay Coloring

The relay coloring procedure uses the idea of sequential coloring algorithm to recolor (i.e., reorganize) the relay candidates. Each neighbor of a relay candidate must belong to one of the color sets resulting from the smallest-last coloring procedure. To describe clearly, we separate the neighbors of a relay candidate into two groups: the neighbors that belong to primary color sets are called “relay neighbors”; the other neighbors are called “non-relay neighbors”. The relay candidates must be selected and ordered according to the amount of their relay neighbors to maximize the adjacency with the nodes of primary color sets. Similar to the degree list illustrated by Figure 2.17, we created a data structure termed “relay degree list” that maps “relay degree” values to lists of nodes (illustrated by Figure 2.25).

[§]The percentage of degree-3 faces of the last four color sets oscillate dramatically since each of them has so few nodes that the total amount of faces is really small.

Relay Degree 5	→	d_1, d_2, \dots, d_{n_5}
Relay Degree 4	→	c_1, c_2, \dots, c_{n_4}
Relay Degree 3	→	b_1, b_2, \dots, b_{n_3}
Relay Degree 2	→	a_1, a_2, \dots, a_{n_2}

Figure 2.25: Relay Degree List

Definition 2.4 The **relay degree** of a relay candidate is the maximum number of its relay neighbors that belong to a common primary color set.

With simple geometric arguments, the number of neighbors with the same color can only be at most five in the plane (illustrated by Figure 2.26(a)). If there were six neighbors in the

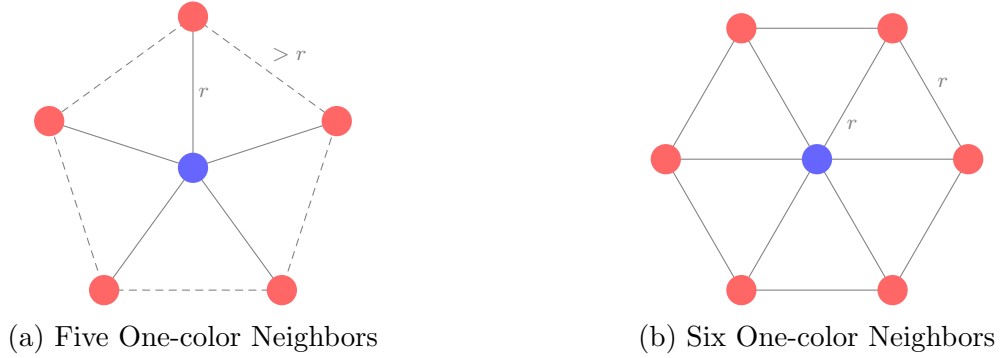


Figure 2.26: Maximum One-color Neighbors

plane with the same color, a hexagon would be formed which contradicts the independency of the color set (as Figure 2.26(b) shows). Note that each node of a relay set should connect to at least two nodes of the paired primary set to relay information, so the relay degree list starts from degree 2 to 5. In order to initialize the list, each relay candidate has a data structure termed “color degree list” that maps color degree values to lists of primary colors (illustrated by Figure 2.27).

Definition 2.5 The **color degree** of a primary color for a relay candidate is the number of the relay neighbors of the candidate node that belong to this primary color set.

The color degree list can be fulfilled by traversing the relay neighbors v_1, v_2, \dots, v_n and putting their corresponding primary colors P_1, P_2, \dots, P_n (where v_i belongs to P_i set) in proper list

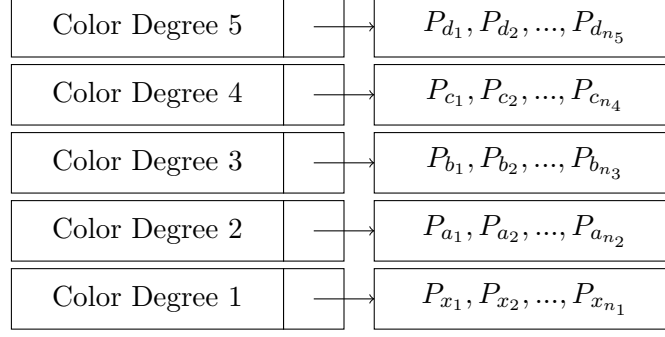


Figure 2.27: Color Degree List

as in bucket sort. In detail, when v_i is traversed, if P_i is not already in the list, it should be appened to the color degree 1 list. If P_i is already in a list, its position in the list should be updated (jumps from its current color degree list to the next higher degree list). Based on Definition 2.4, the relay degree of a relay candidate is determined by the largest degree value in its color degree list that corresponds to a non-empty color list. In other words, the relay degree of the relay candidate represents the primary colors with the maximum color degree for the node. Thus, the relay degree list can be fulfilled by computing the relay degree of each relay candidate. Drawing from our experience in constructing the degree list for the smallest-last ordering algorithm, we can use either an array of doubly linked lists or a hash map to construct both the relay degree list and the color degree list. This enables us to achieve a time complexity of only $O(1)$ for updating the two lists.

The greedy coloring procedure is based on a dynamic sequence resulting from the relay degree list. For the preparation, a list of relay colors C_1, C_2, \dots, C_n is generated by traversing the selected primary colors P_1, P_2, \dots, P_n where C_i is paired with P_i [¶]. The coloring is an iterative trial process illustrated by the while loop in the pseudocode of Algorithm 2. Each iteration removes a primary color P from the corresponding color list of the node u that has maximum relay degree, and tries to greedy-color u with the relay color C that is paired with P . If the trial process failed, then the relay degree of u would be updated (i.e., its position in the relay degree list might be changed) because of the removal of P . There are

[¶]In our program, each relay color value C_i is calculated by adding a fixed offset k to the paired primary color value P_i where k is the number of color sets resulting from the smallest-last coloring algorithm.

Algorithm 2 Relay Coloring

```
1: function RLColor(Candidate Set  $V_R$ , Primary Set  $P^*$ )
2:    $S \leftarrow$  new List;
3:    $D \leftarrow$  new Relay-Degree-List( $V_R$ ,  $P^*$ );
4:   Prepare a list of relay colors  $C^*$  paired with  $P^*$ ;
5:   while  $D \neq \emptyset$  do ▷ trial process of greedy-color  $u$ 
6:     Remove a largest relay degree node  $u$  from  $D$ ;
7:     Pick a relay color  $C$  from  $C^*$  specified by  $u$ ;
8:     if  $C$  was assigned to any non-relay neighbors then
9:       Update the relay degree of  $u$ ;
10:      if the relay degree of  $u \geq 2$  then
11:        Put  $u$  back to  $D$ ;
12:      else
13:        Assign  $C$  to  $u$ ;
14:         $S \leftarrow S + u$ ;
15:   return  $S$ ;
```

two situations that would cause the complete removal of u from the relay degree list:

- the updated relay degree is less than 2;
- u has been successfully greedy-colored.

Once the relay degree list becomes empty, the greedy coloring procedure ends. In conclusion, both the initialization of the relay degree list and the greedy coloring procedure have time complexity of $O(|V_R| + |E_{G-R}|)$ where V_R represents the relay candidate set and E_{G-R} represents the set of edges between the relay candidates and the nodes of primary color sets. Figure 2.28 shows the relay coloring procedure of our sample RGG. Table 2.12 presents

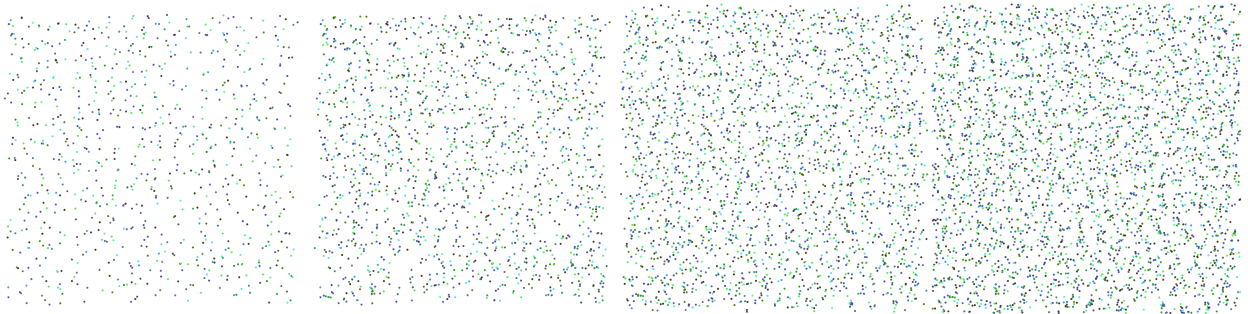


Figure 2.28: Relay Coloring Procedure[†]

screenshots to compare the relay coloring procedure applied on four distinct surfaces. Fig-

Table 2.12: Relay Coloring

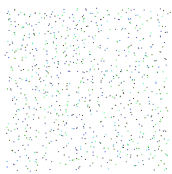
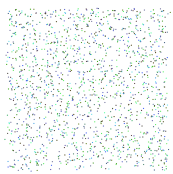
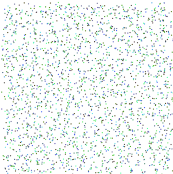
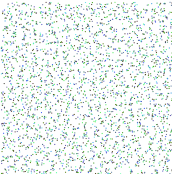
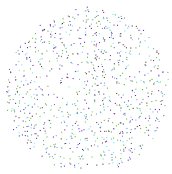
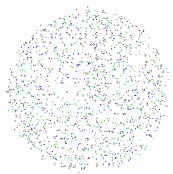
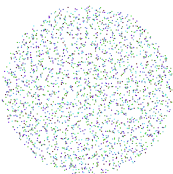
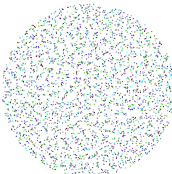
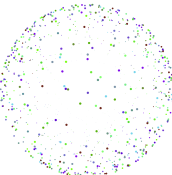
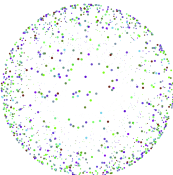
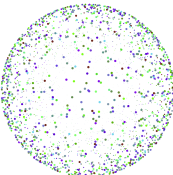
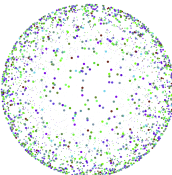

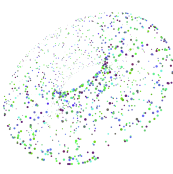
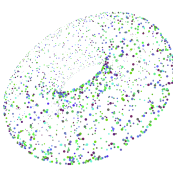
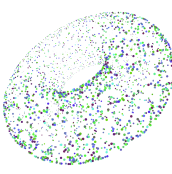
Geometry	25%	50%	75%	100%
Unit Square $G(10^4, 0.057)$				
Unit Disk $G(10^4, 0.101)$				
Unit Sphere $G(10^4, 0.201)$				
Unit Torus $G(10^4, 0.252)$				

Figure 2.29 shows the color-size plot resulting from the relay coloring procedure. The relay

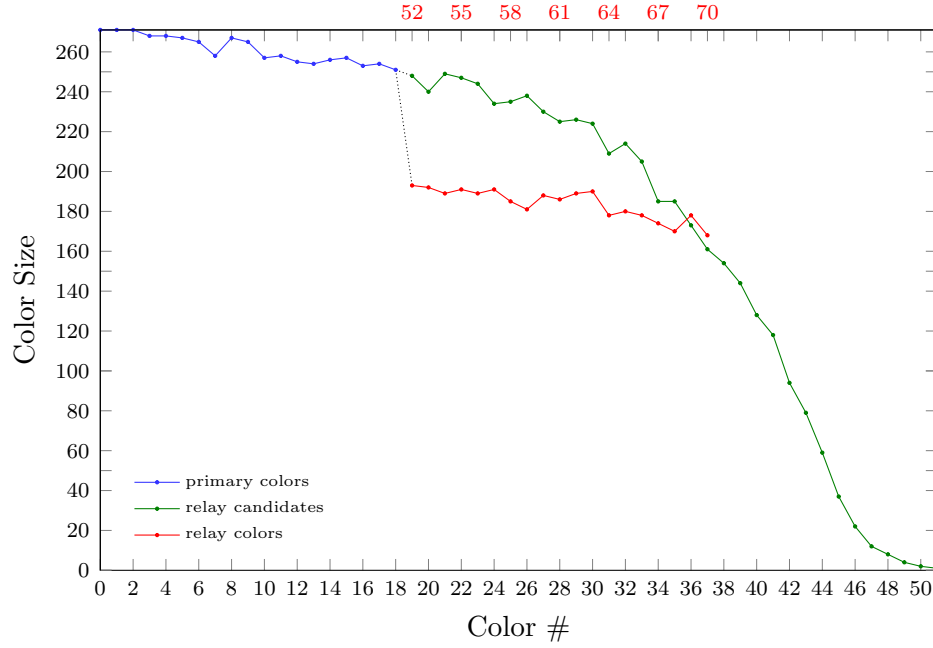


Figure 2.29: Color Size Plot of Relay Coloring

colors (i.e., red plot) are one-to-one correspondent to the primary colors (i.e., blue plot) with a similar size gap. Figure 2.30 shows Gabriel graphs and the percentage of degree-three faces $|f(3)|$ from evenly selected four sample relay color sets of our sample RGG. Most nodes of

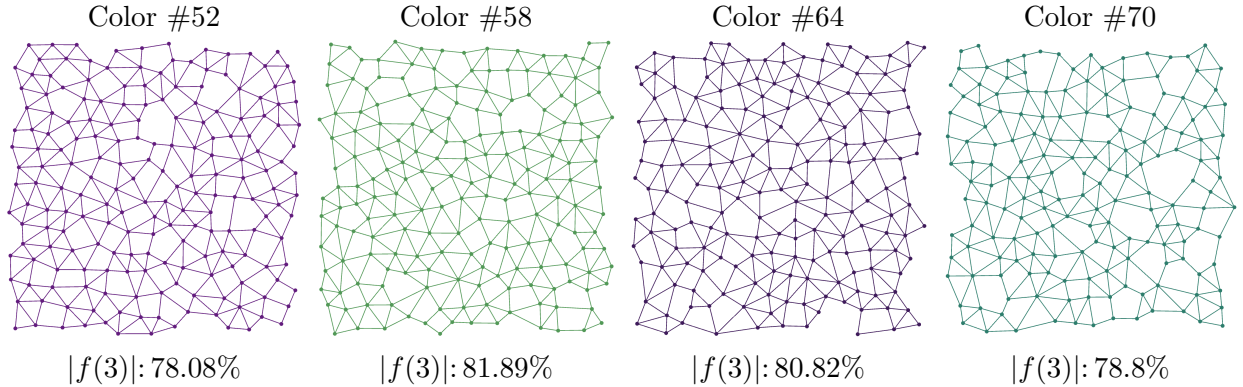


Figure 2.30: Degree-3 Faces of Selected Relay Color Sets with Gabriel Rules

relay color sets are still close-packed in triangular lattice.

Note that not all relay candidates were colored in the relay coloring procedure. The set of nodes who failed the greedy-coloring trial process are termed “surplus”, and Definition 2.6

provides a more general-purpose definition.

Definition 2.6 *Surplus* is the set of nodes that are not used by any resulting backbones.

There are 1544 (15.44%) surplus nodes after the relay coloring procedure of our sample RGG. Now the RGG has been divided into three groups of nodes by the smallest-last coloring and relay coloring procedures as shown by Figure 2.31. Table 2.13 presents screenshots to

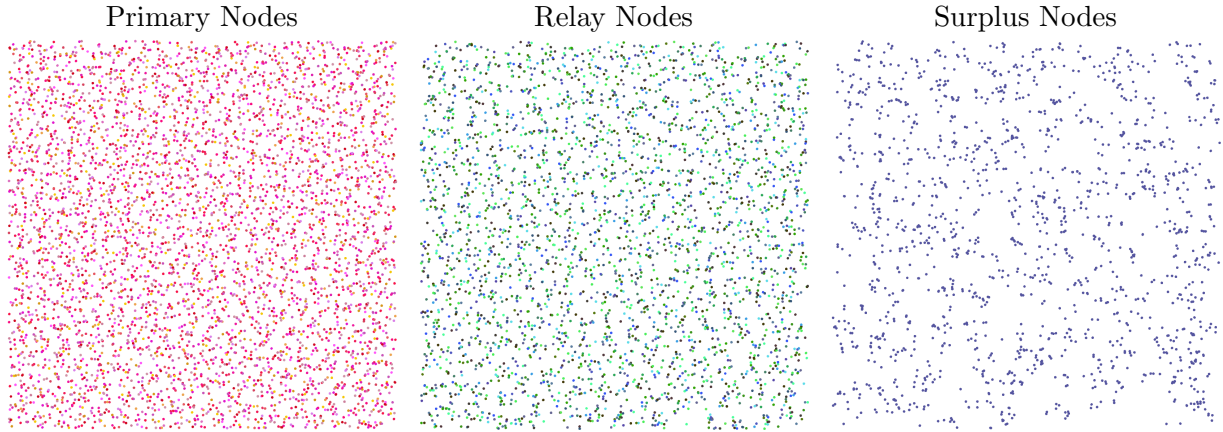
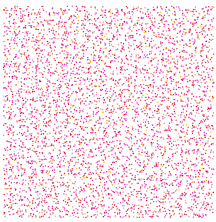
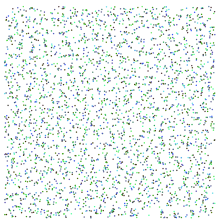
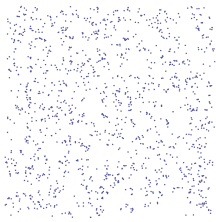
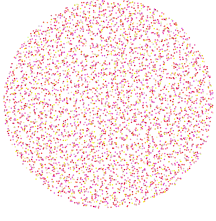
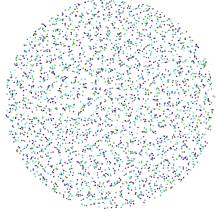
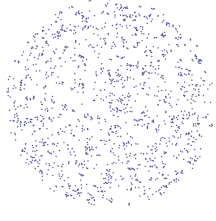
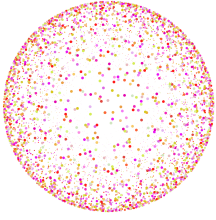
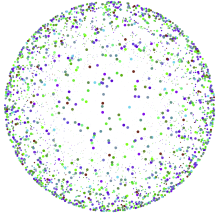
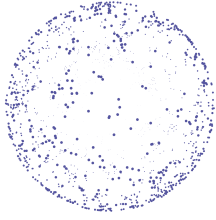
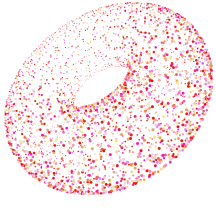
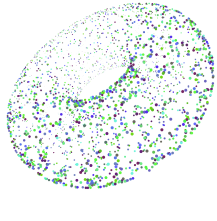
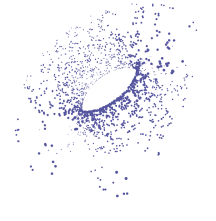


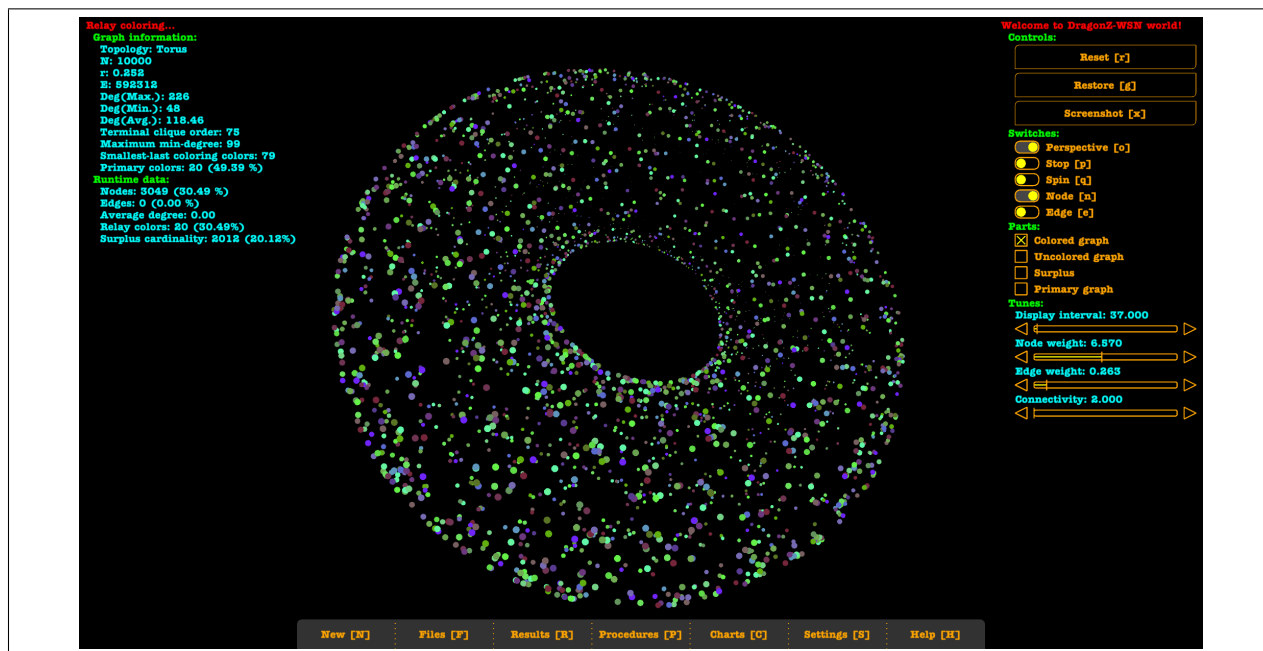
Figure 2.31: Nodes after Smallest-last and Relay Coloring

compare the resulting groups of nodes after the two-phase coloring algorithm applied on the four distinct surfaces. Video 2.5 demonstrates the relay coloring animation with resulting color sets and surplus nodes for these four surfaces. The initial two resulting relay color sets of each surface will also be applied Gabriel rules to show the density of triangular faces. The video also shows the color-size plots after relay coloring for all surfaces. Figure 2.32 presents the plots of the percentage of the degree-3 faces for both primary and relay color sets resulting from the two-phase coloring algorithm applied on our sample RGG. All primary sets have around 90% of faces which have degree 3, while all relay sets have around 80% of faces which have degree 3. Therefore, as we sought for, the resulting paired primary and relay set forms a subgraph that approximates the bi-regular three and four lattice grid. Figure 2.33 shows the evenly selected four bipartite subgraphs from the sample RGG.

2.4.4. Backbone Refinement

Table 2.13: Nodes after Smallest-last and Relay Coloring

Geometry	Primary Nodes	Relay Nodes	Surplus Nodes
Unit Square $G(10^4, 0.057)$			
Unit Disk $G(10^4, 0.101)$			
Unit Sphere $G(10^4, 0.201)$			
Unit Torus $G(10^4, 0.252)$			



Video 2.5: Relay Coloring

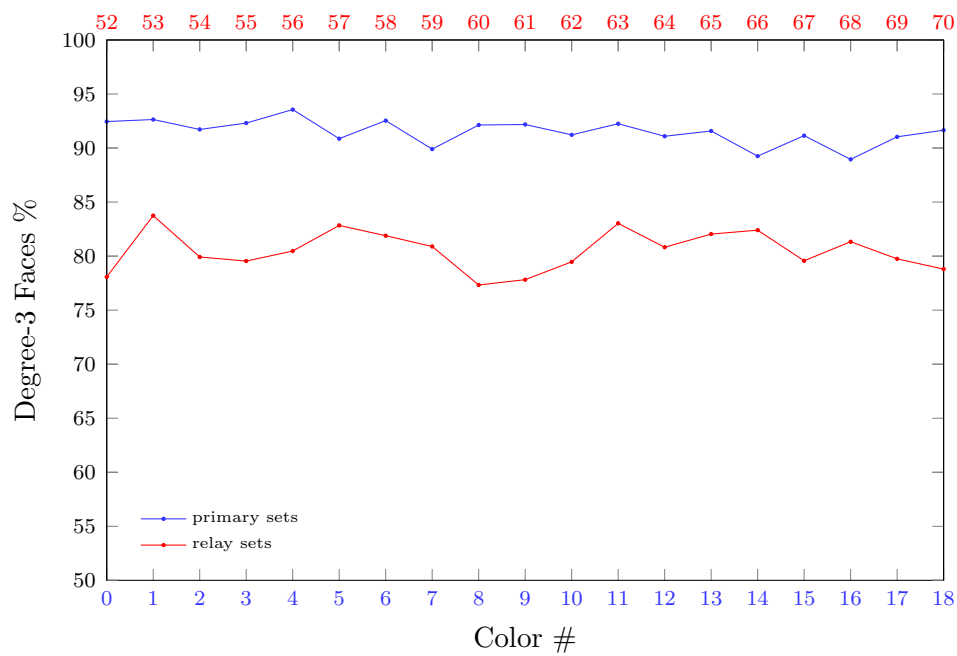


Figure 2.32: Degree-3 Face Percentage of Primary and Relay Color Sets

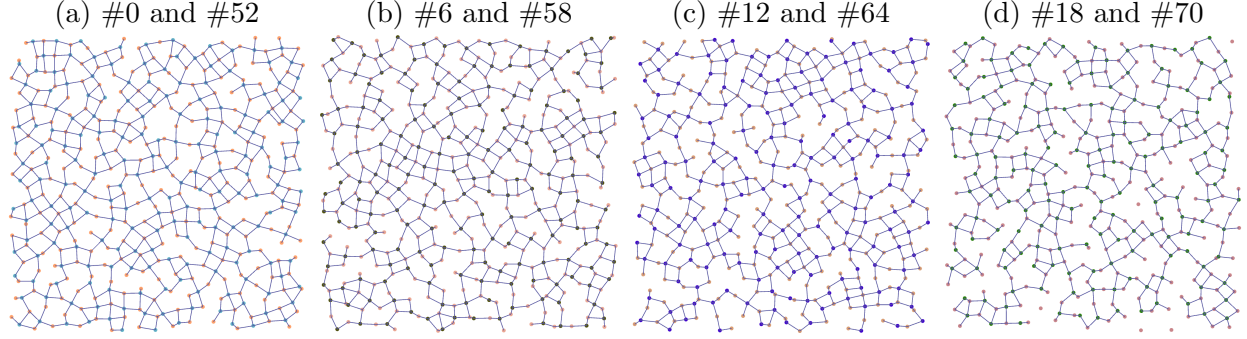


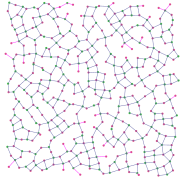
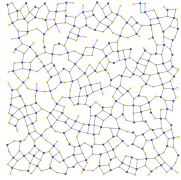
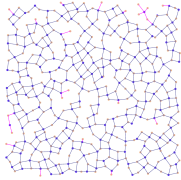
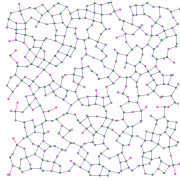
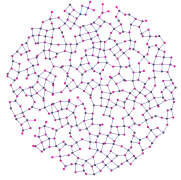
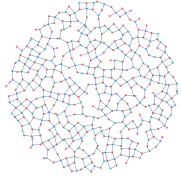
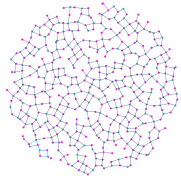
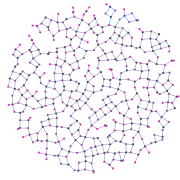
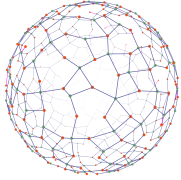
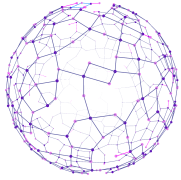
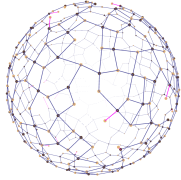
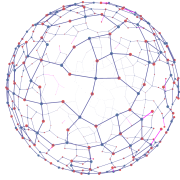
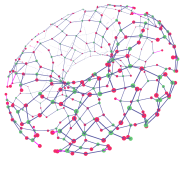
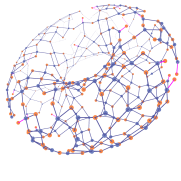
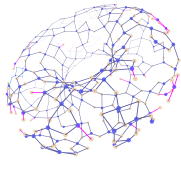
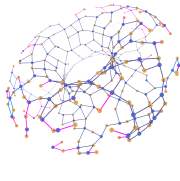
Figure 2.33: Sample Paired Bipartite Subgraphs

By far the bipartite subgraphs given by the paired primary and relay sets resulting from the two-phase coloring algorithm are still not fully qualified as virtual backbones, since some of them might not be fully connected. Figure 2.33 (d) shows such an example, which has five singletons (isolated nodes) and a component. Connected components can be easily determined by Depth First Search (DFS) algorithm in linear time and it is preferred to keep the largest component (termed “giant component”) as the virtual backbone by removing all other smaller components (termed “minor components”). Therefore, we have the following definition of virtual backbones resulting from our partitioning algorithms applied on an RGG.

Definition 2.7 *For a WSN modeled as an RGG, the **virtual backbones** are the collection of giant components of the bipartite subgraphs each is composed of a paired primary and relay set resulting from the two-phase sequential coloring algorithm.*

Among the 19 paired primary and relay sets given by the coloring algorithm of our sample RGG, only three bipartite subgraphs have minor components (circled out in Figure 2.34). According to Definition 2.6, the nodes of these minor components to be removed also belong to the surplus set, so our sample RGG finally generates 1552 surplus nodes. In other words, 84.48% of the nodes belonging to the WSN have been used in this BCP while the remaining 15.52% do not participate in the routing or other in-network services (they only collect data of the region as slave nodes). Table 2.14 presents screenshots to compare evenly selected sample backbones for the four surfaces.

Table 2.14: Evenly Selected Sample Backbones

Backbone	1	2	3	4
Domination	100%	100%	100%	99.93%
Unit Square $G(10^4, 0.057)$				
3-coverage	95.10%	96.93%	96.65%	94.96%
Domination	100%	100%	99.99%	99.87%
Unit Disk $G(10^4, 0.101)$				
3-coverage	96.40%	97.23%	96.52%	93.83%
Domination	100%	99.99%	99.99%	100%
Unit Sphere $G(10^4, 0.201)$				
3-coverage	99.15%	98.04%	98.48%	97.78%
Domination	99.94%	99.99%	99.58%	92.17%
Unit Torus $G(10^4, 0.252)$				
3-coverage	97.27%	96.59%	94.09%	78.77%

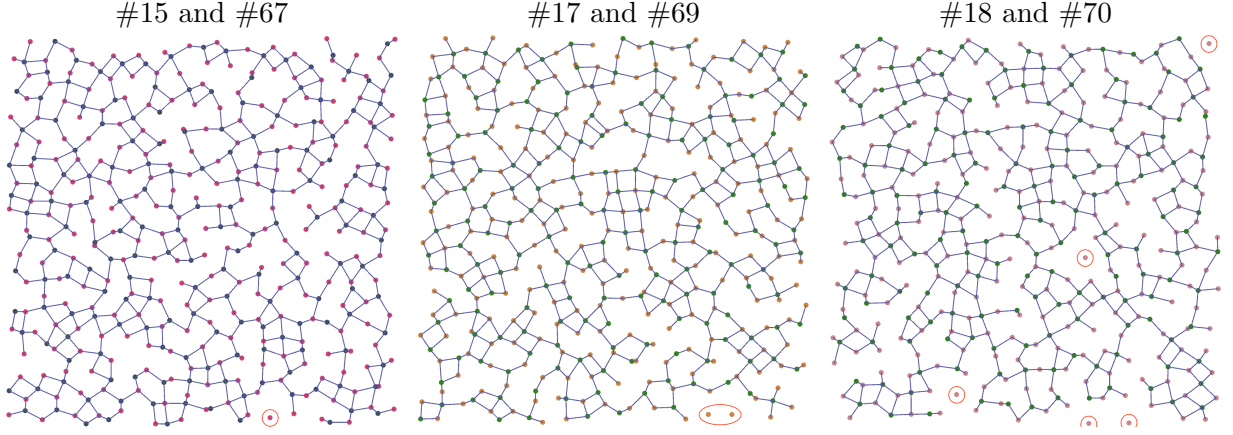


Figure 2.34: Sample Bipartites with Minor Components

2.4.4.1. Performance Metrics

We mainly consider two topological performance metrics for each virtual backbone: domination and k -coverage. Domination refers to how many nodes of the RGG are covered by the backbone, while k -coverage means that the nodes of the RGG are covered by at least how many (k) backbone nodes. Specifically, for the k -coverage metric, we care more about how many nodes are covered by at least three backbone nodes ($k = 3$) for better localization and synchronization [31, 60]. Therefore, both metrics can be represented by the percentage of nodes that meet the desired property. Particularly, according to Definition 2.1, in the case of cluster-based control structure, the nodes of the primary set are used as cluster-heads and the nodes of the relay set are used as routing relays. In this sense, the domination of a backbone is the domination of the primary set, so besides the domination of a whole backbone network (Backbone Domination), we also measure the domination of its primary set (Primary Domination). Table 2.15 shows these performance metrics in percentage for all the 19 backbones (giant components) of our sample RGG.

2.4.4.2. Robustness of Backbones

A virtual backbone needs to be a connected network in order to successfully forward data. Although the backbone defined by the giant component in Definition 2.7 is indeed a connected subgraph, some concerns for the robustness may arise since sensors could fail at

Table 2.15: Domination and 3-coverage Percentage

Backbone	Primary Domination	Backbone Domination	3-Coverage
Color #15 & #67	99.95%	99.96%	95.29%
Color #17 & #69	99.51%	99.53%	93.75%
Color #18 & #70	99.43%	99.57%	91.44%
Color #11 & #63	99.98%	100%	95.58%
Color #1 & #53	100%	100%	97.20%
Color #0 & #52	100%	100%	97.35%
Color #7 & #59	99.99%	99.99%	95.24%
Color #5 & #57	100%	100%	97.14%
Color #3 & #55	99.99%	100%	97.03%
Color #10 & #62	99.95%	100%	95.90%
Color #9 & #61	99.95%	99.98%	95.99%
Color #6 & #58	99.99%	100%	96.38%
Color #14 & #66	99.97%	99.99%	94.34%
Color #4 & #56	100%	100%	96.77%
Color #8 & #60	100%	100%	96.39%
Color #2 & #54	100%	100%	96.89%
Color #13 & #65	99.98%	100%	95.13%
Color #12 & #64	99.94%	100%	94.83%
Color #16 & #68	99.97%	99.98%	94.19%

random times in an unattended network. Topologically, two structures (tail and cut-node) are not preferred in a virtual backbone requiring strong fault tolerance.

- A **tail** is a visual structure concept, which is the tree structure part of a connected graph. From the algorithmic point of view, tails can be obtained by continuously removing a degree-one node from the remaining graph until all remaining nodes have degrees larger than or equal to two. The removed nodes are the tails of the graph and we term the remaining graph (with no tails) as “two-core”. Tail structures will lower the reliability of a backbone network since any node failures of a tail will disconnect the backbone. Algorithm 3 will remove tails from a graph in linear time $O(|V|+|E|)$ (where V is the node set and E is the edge set) by using the degree list data structure (shown in Figure 2.17) mentioned in the smallest-last ordering algorithm.

Algorithm 3 Removing Tails from Graph

```

1: function TWO-CORE(Graph  $G$ )
2:   Initialize a degree list  $D$  from  $G$ ;
3:   while degree-1 list  $D_1 \neq \emptyset$  do
4:     Remove a node  $u$  from  $D_1$ ;
5:     Update all neighbors of  $u$  in  $D$ ;
6:   return  $D$ ;

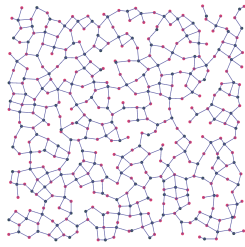
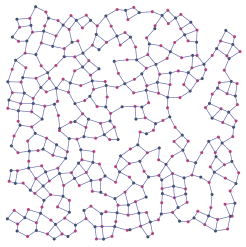
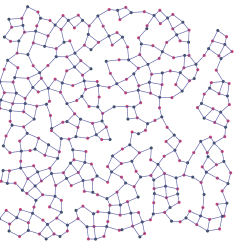
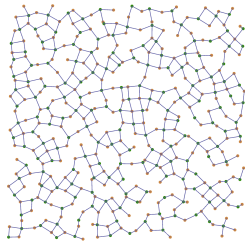
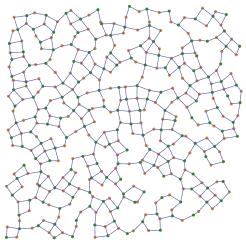
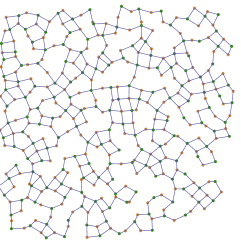
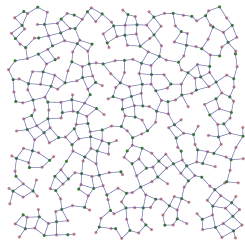
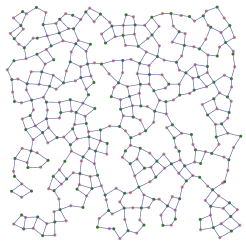
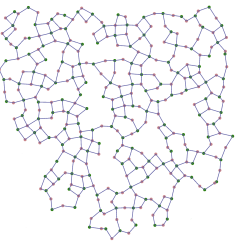
```

▷ D contains nodes other than trails

- In graph theory, a **cut-node** refers to any node that, upon its removal, leads to an increase in the number of connected components within the graph. By this definition, every node of a tail can be considered a cut-node. However, even if all tails were removed from a graph, there might still exist cut-nodes (e.g., the end nodes of a bridge). A maximal subgraph with no cut-node is called a “block” of the graph. To make a backbone network more robust, we can keep the largest block (termed “giant block”) by removing all other smaller blocks (termed “minor blocks”). A linear time algorithm based on DFS to determine blocks (nonseparable components) of a graph is provided in [27]. The algorithm relies on the order of each node in the DFS traversal of a graph. Each node u introduces a new property termed “lowpoint” which is the smallest order of a node v that can be reached from u by a directed path with no cycle,

followed by at most one back edge. It is proved that if $u.order > 1$ and $v.lowpoint \geq u.order$, then u is a cut-node. Algorithm 4 is a pseudo-code of determining the giant block of a graph that has time complexity $O(|E|)$ where E is the edge set of the graph. Table 2.16 shows the three types (giant component, two-core and giant block) of the backbones from the samples in Figure 2.34.

Table 2.16: Sample Backbones of Different Robustness

Backbone	Giant Component	Two-core	Giant Block
Color Pair #15 #67			
Color Pair #17 #69			
Color Pair #18 #70			

The levels of robustness provided by the three types of backbone (giant component \leq two-core \leq giant block) is obtained by removing nodes of unreliable structure. So the more robust a virtual backbone is, the lower coverage metrics it would have. The removal of unreliable nodes also increases the amount of surplus nodes. The surplus nodes of the three backbone types

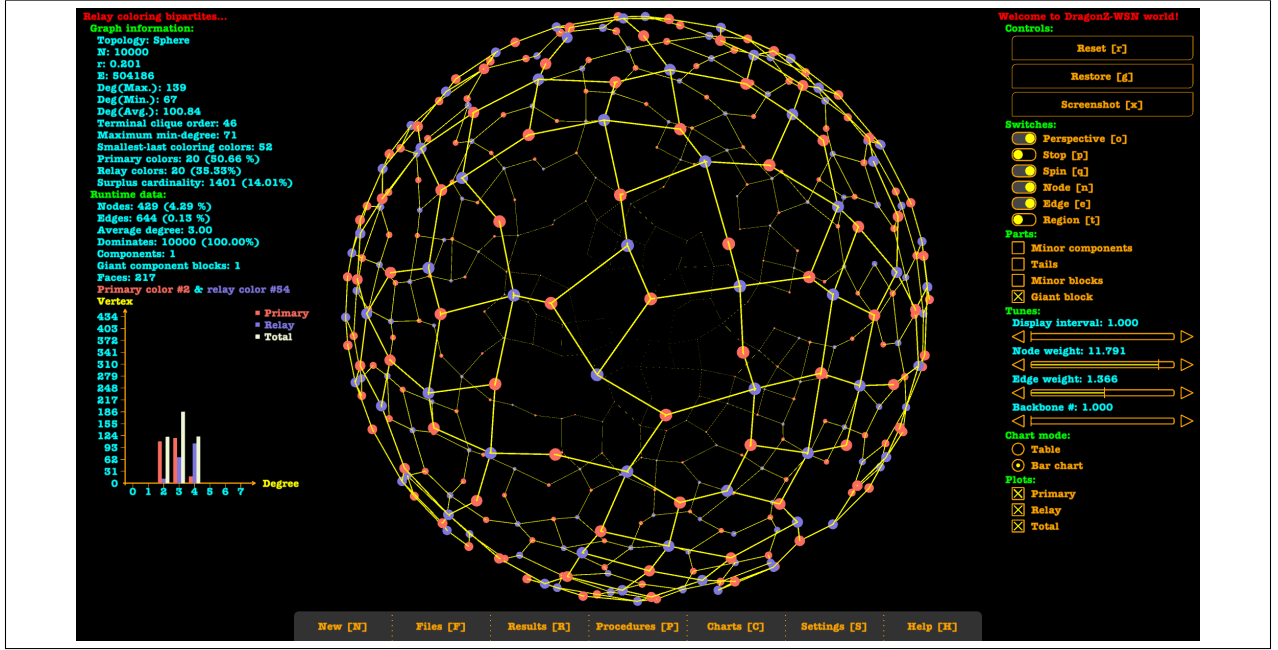
Algorithm 4 Determine Giant Block from Graph

```
1:  $i \leftarrow 0$ ; ▷ global order counter
2:  $T \leftarrow \text{new Stack}$ ;

3: function GIANT-BLOCK(Graph  $G$ )
4:    $B \leftarrow \emptyset$ ; ▷ store giant block
5:    $s \leftarrow \text{a start node of } G$ ;
6:   for each Node  $u$  and Edge  $e$  of  $G$  do
7:      $u.\text{order} \leftarrow 0$ ;
8:      $e.\text{mark} \leftarrow \text{false}$ ; ▷ mark as not traversed
9:   DFS( $B, s, \text{null}$ ); ▷ the predecessor of  $s$  is null
10:  return  $B$ ;

11: procedure DFS(Block  $B$ , Node  $u$ , Node  $p$ )
12:  if  $u.\text{order}=0$  then
13:     $i \leftarrow i + 1$ ;
14:     $u.\text{lowpoint} \leftarrow u.\text{order} \leftarrow i$ ;
15:     $T.\text{push}(u)$ ;
16:    while Edge  $e(u, v).\text{mark} = \text{false}$  do
17:       $e.\text{mark} \leftarrow \text{true}$ ; ▷ mark as traversed
18:      if  $v \neq p$  then
19:        DFS( $B, v, u$ );
20:      if  $p \neq \text{null}$  then
21:        if  $u.\text{lowpoint} < p.\text{order}$  then
22:           $p.\text{lowpoint} \leftarrow \min(p.\text{lowpoint}, u.\text{lowpoint})$ ;
23:        else
24:           $C \leftarrow \text{new List}$ ; ▷ store a determined block
25:          repeat
26:             $v \leftarrow T.\text{pop}()$ ;
27:             $C.\text{add}(v)$ ;
28:          until  $v \neq u$ 
29:           $C.\text{add}(p)$ ; ▷ new determined block  $C$ 
30:          if  $C.\text{size}() > B.\text{size}()$  then
31:             $B \leftarrow C$ ;
32:      else
33:         $p.\text{lowpoint} \leftarrow \min(p.\text{lowpoint}, u.\text{order})$ ;
```

from our sample RGG are: 1552 (15.52%), 2103 (21.03%) and 2779 (27.79%) respectively. Therefore, it is preferred to adjust the robustness of backbones according to the application requirements of the WSN. Figure 2.35 shows the domination metrics of different type of backbones of samples in Figure 2.34. Finally, Video 2.6 shows all the resulting bipartite subgraphs of the four surfaces listed in Table 2.14. It demonstrates the three types (giant



Video 2.6: Backbone Refinement

component, two-core and giant block) of virtual backbones for each surface.

2.4.4.3. Other Adjustments

Number of Backbones According to Definition 2.3 and 2.7, the number of virtual backbones resulting from the two-phase sequential coloring algorithm is adjustable by the number k of selected primary sets. The number of generated virtual backbones typically equals to k , but if k is too large, there will not be enough relay candidates to form paired relay sets. Figure 2.36 shows the number of backbones of k selected primary sets of our sample RGG. Ideally, we expect the size s of surplus set to be as small as possible, in other words, nearly

^{||}The program running results may be slightly different each time caused by randomness.

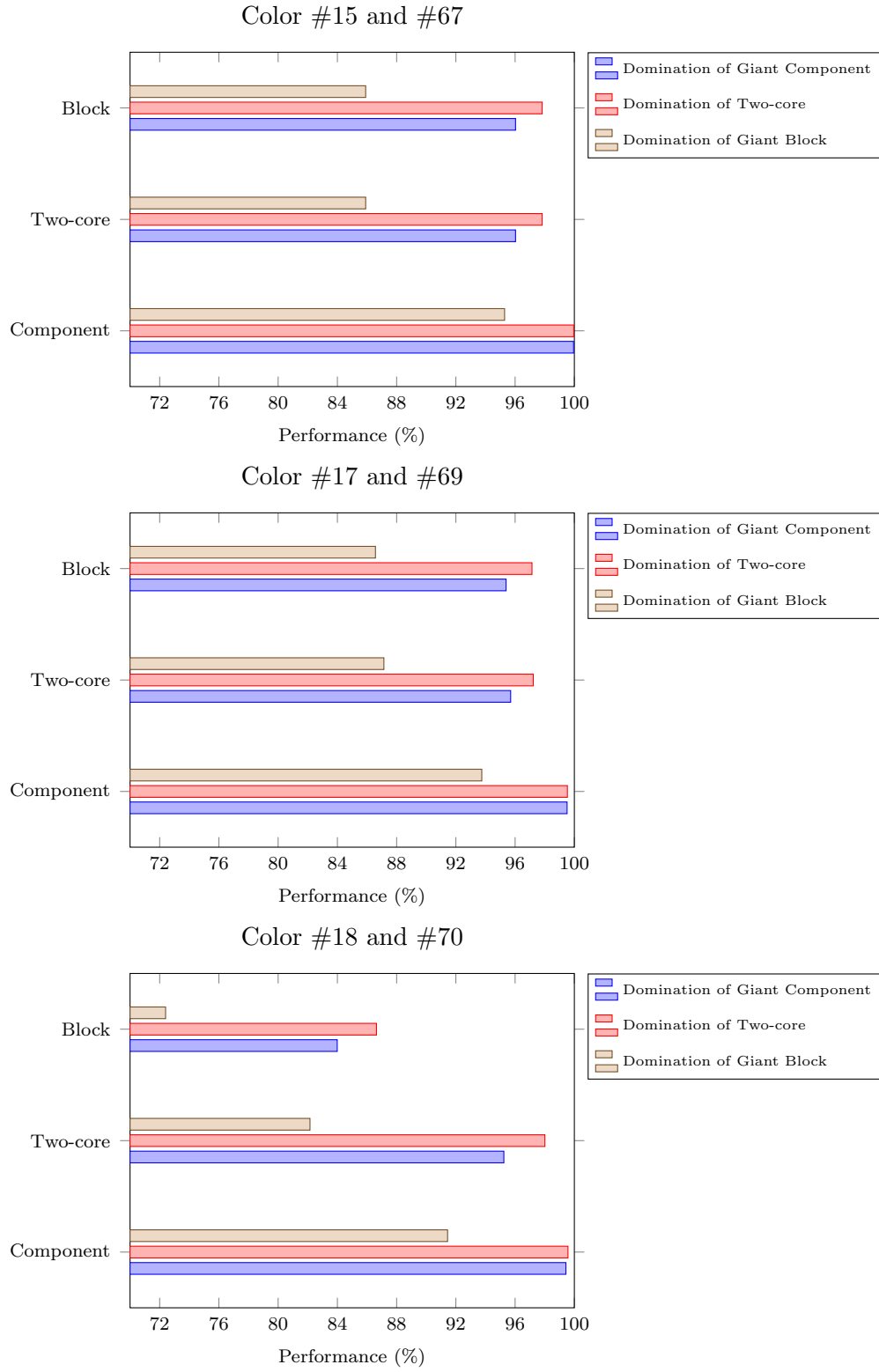
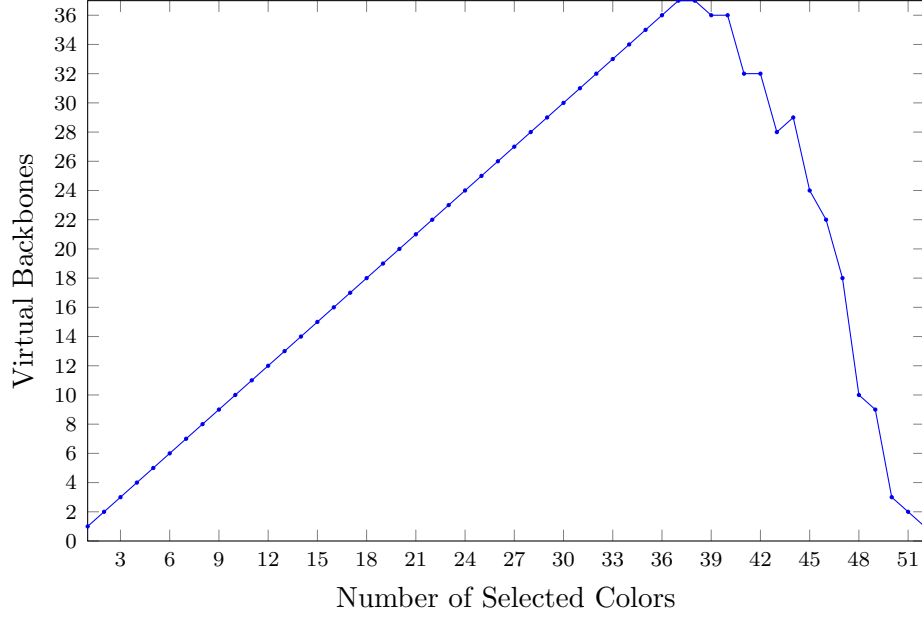


Figure 2.35: Domination and 3-coverage of Sample Backbones of Different Types



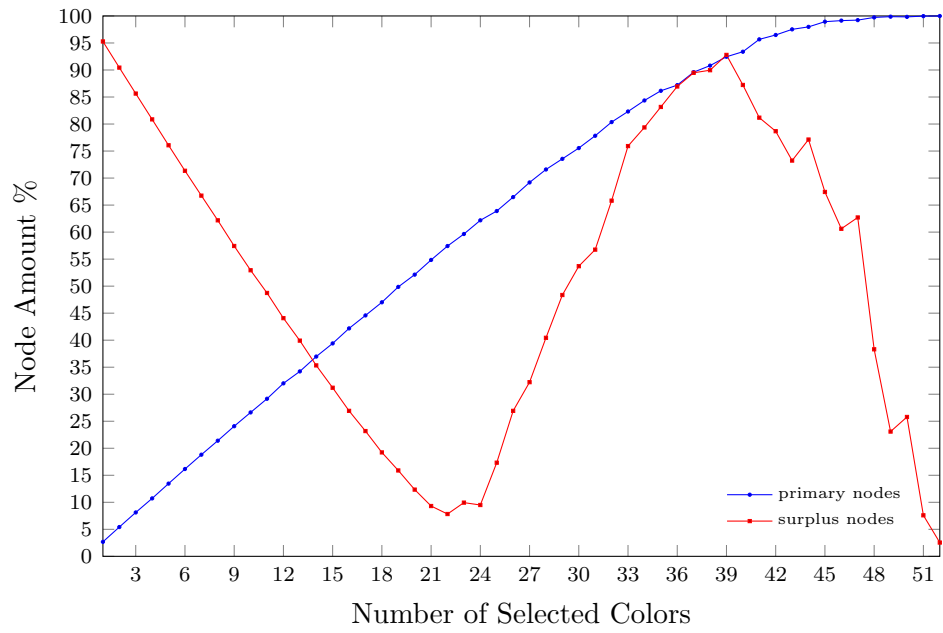


Figure 2.37: Surplus Nodes vs. Selected Primary Nodes

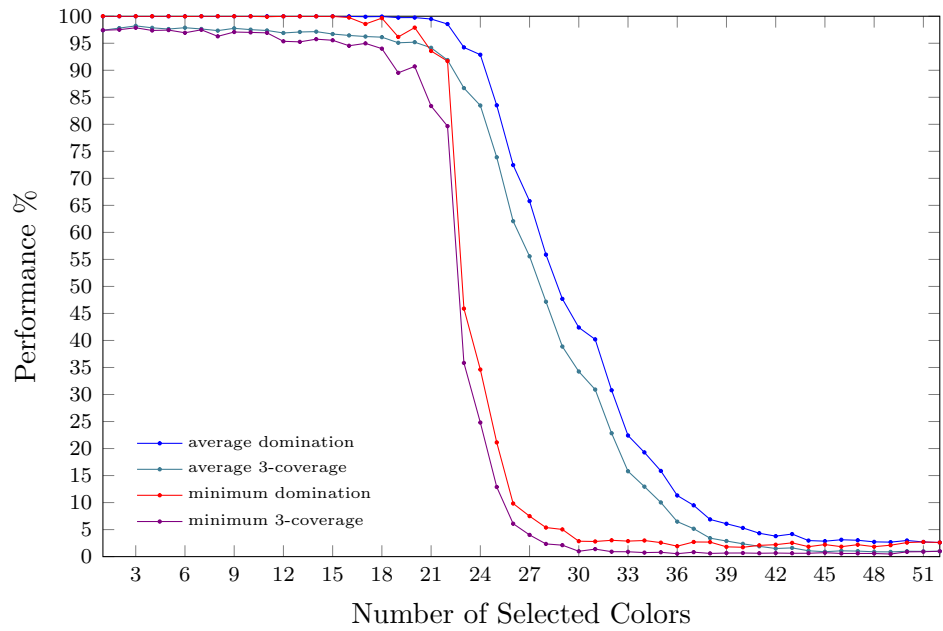


Figure 2.38: Performance of k Selected Primary Colors

since each node of a relay set needs to connect to at least two nodes of the paired primary set to relay information. If any application requires a stronger connectivity, the lowest degree value of the relay degree list can be increased to fulfill the requirement. However, stronger connectivity may lower the overall performance of generated backbones and also increase the amount of surplus nodes. By selecting 19 primary color sets of our sample RGG, Figures 2.39 and 2.40 show the performance metrics and surplus nodes based on different connectivities.

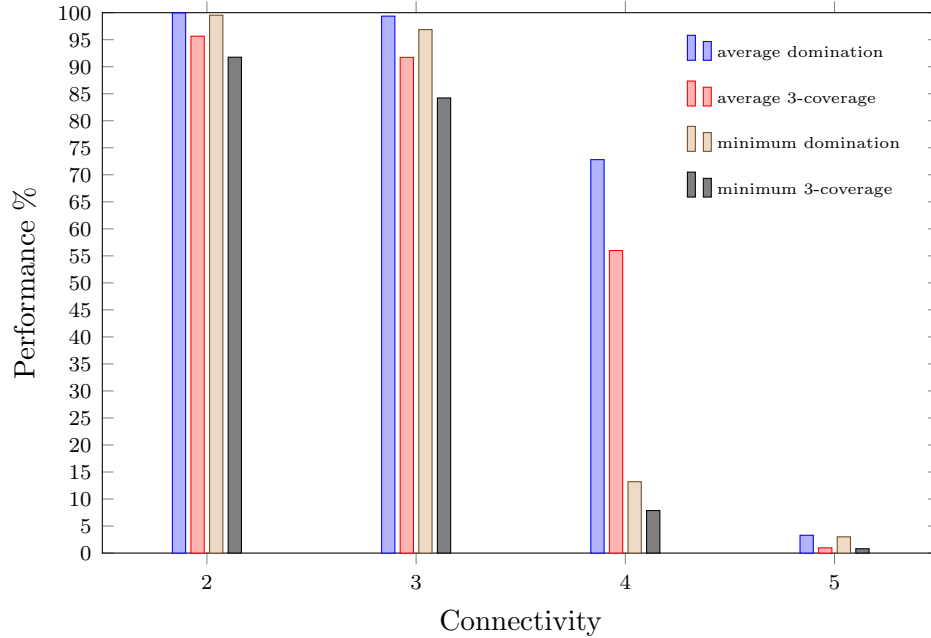


Figure 2.39: Performances of Different Connectivities

It's obvious that connectivity 5 generates less backbones of really low performance and large size of surplus set since the maximum possible relay degree is 5 on planar surface according to the geometric arguments (illustrated by Figure 2.26). Figure 2.41 shows sample backbones of different connectivities.

2.5. Conclusion

A backbone formation method of partitioning a Wireless Sensor Network (WSN) modeled as a Random Geometric Graph (RGG) based on a two-phase sequential coloring algorithm is proposed. The concept of partitioning a WSN into a collection of disjoint backbone grids

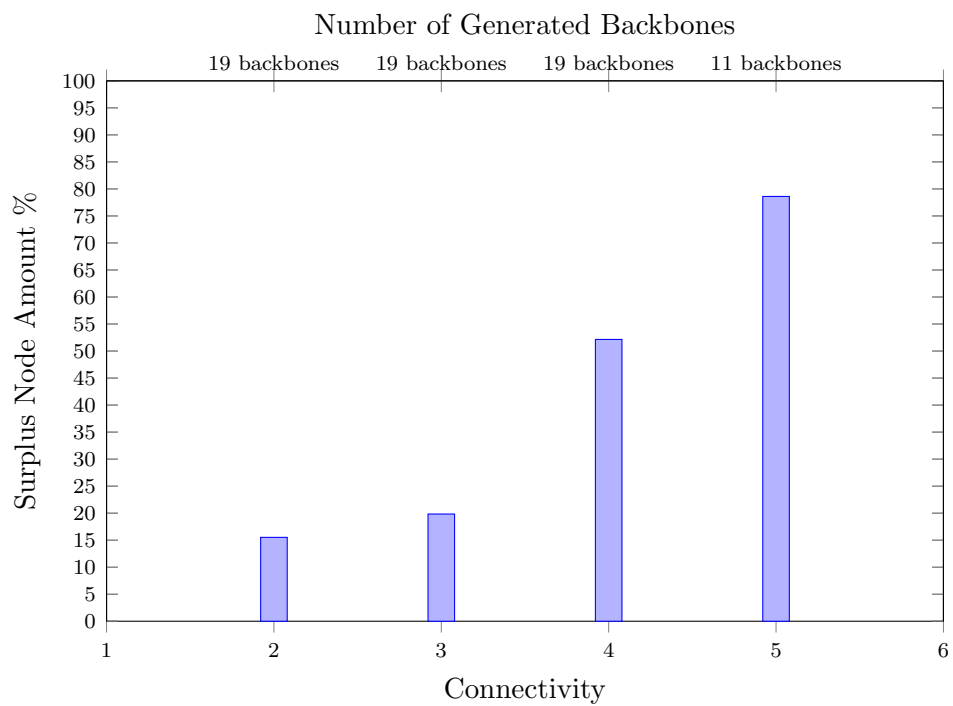


Figure 2.40: Surplus Nodes of Different Connectivities

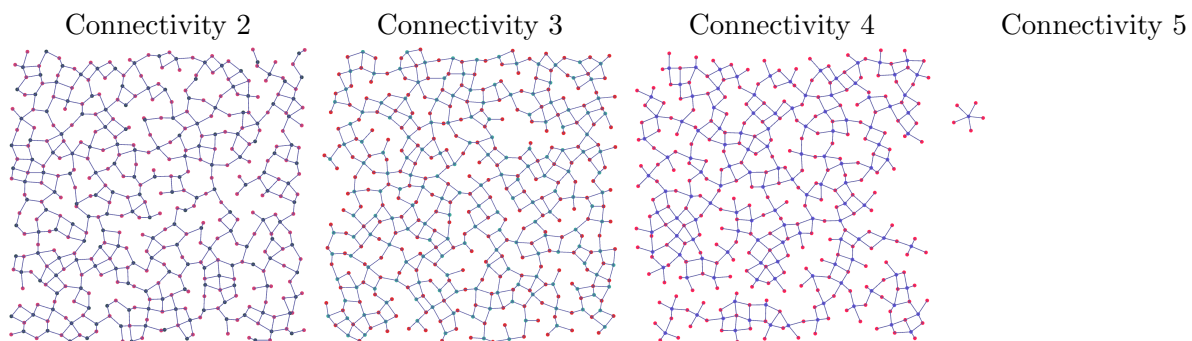


Figure 2.41: Sample Backbones of Different Connectivities

(instead of forming a single optimal backbone) is termed Bipartite Component Partition (BCP) whose activity can be duty-cycled/rotated/scheduled to offer better fault-tolerance, load-balancing, and scalability. Each backbone grid is composed of a primary independent set and a paired relay independent set to transfer collected information from sensors to sink nodes through a cluster-based structure of the WSN. BCP also prolongs the network lifetime and raises the possibility of catering to several quality of service requirements.

2.5.1. Contributions of Visualized Algorithm Engineering (VAE)

Visualization plays a critical role in this research which assisted us in finding patterns of network topologies and hence inspired the new algorithm. A 3D program was built to implement the algorithmic procedures via a programming language “Processing” that is popular in visual arts area [62]. This chapter only presents four geometries (unit square, unit disk, unit sphere and unit torus) as models of WSN. Other geometries such as unit triangle and Klein bottle were also implemented (as Figure 2.42 shows), which not only emphasizes that our algorithms are topology-based, but also demonstrates the designing power of our 3D program. In order to implement the visualized algorithm engineering work,

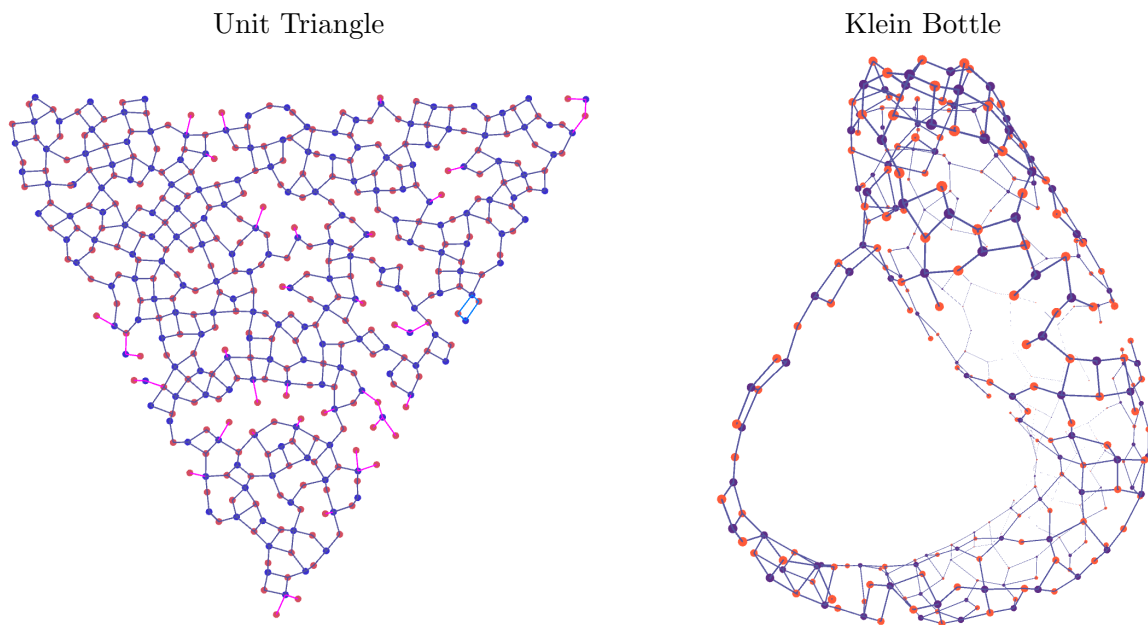


Figure 2.42: Sample Backbones of Other Geometries

two efficient algorithms (sweep method and cell method) for constructing large sparse RGGs are introduced as byproducts of our research. The visual display of our two-phase coloring algorithm for backbone determination is proposed as a valuable teaching tool for devising further backbone algorithms. The full extent of the visual effects produced by our program cannot be conveyed through this two-dimensional paper print medium, and the pre-recorded videos can only demonstrate a portion of the effects as they are constrained by the limitations of their format.

2.5.2. Evaluation and Future Works

Generally, the formation techniques of WSNs are classified into distributed networks and centralized networks. In the former technique, nodes are autonomous and the communication is only between neighboring nodes while, for the latter, the network formation is controlled by a single device. The backbone determination method discussed in this paper is based on the WSNs of cluster-control structure which belongs to the distributed formation techniques [13]. The main characteristics of distributed networks include the following:

- (a) There are autonomous devices.
- (b) Each node shares information with its neighborhood.
- (c) It is suitable for distributed applications (multi-agent systems, self-organized systems, etc.)
- (d) The information is mainly forwarded to a single node.
- (e) Interconnection devices (routers, bridges, etc.) are not required.
- (f) Their flexibility allows targeting harsh environments.

Apart from the formation of the WSN, our backbone determination algorithms are all distributed non-localized algorithms. A distributed algorithm is called “localized” if each node decides on its own behavior based on only the information from nodes within a limited hop

distance. In contrast, a distributed non-localized algorithm requires nodes to acquire global knowledge of the network to perform correct operations [15]. Therefore, even though the WSN itself is a distributed network, global adjacency information of all nodes is still required to establish degree list and relay degree list for performing the smallest-last coloring and relay coloring procedures to determine the BCP. If a pure autonomous distributed setting is required, the two-phase sequential coloring algorithms need to be replaced by two-phase localized, geometry-oblivious coloring algorithms (such as [6]), but the backbone formation method of selecting primary sets and paired relay sets is still the same.

Although our backbone determination algorithm requires a station tower or a Global Positioning System (GPS) to acquire global knowledge (adjacency information) of all sensor nodes, it has the following advantages:

- **Efficiency** — All procedures (including the construction of an RGG to model a WSN) have linear time efficiency.
- **Robustness** — Multi-partitioning method of generating a collection of backbone grids not only prolongs the network longevity, but also enhances the robustness (any single backbone failed, there are other backbones available in the collection). Besides, we analyzed and provided several ways of adjusting the robustness to fit different application requirements.
- **Topology-base** — Topology-based algorithms allow us to apply the backbone determination procedure to regions of different geometries (demonstrated by our own visual program). Adjacency information of each node is the only required information to perform the algorithms.
- **Performance** — Topologically, random deployment has a low transfer delay [58], and our bi-regular three and four lattice structure has been shown to have great k -coverage performance. Smallest-last coloring has been shown to generate a collection of similar sized disjoint and verifiably dominating sets that have equally good performance in

covering the network [47, 48]. Relay coloring ensures the paired relay sets to maintain maximum connectivities with the selected primary sets.

Therefore, generally speaking, our BCP method is suitable for applications that require large amount of sensors and have access to the adjacency information of global sensors (such as planetary explorations [19, 59]).

Suggested future works include: comparison of the properties of the generated backbones with those obtained from other backbone determination algorithms on a common network dataset; research on characteristics of generated backbones on other 3D solid geometries (such as the unit cube and unit ball implemented in our 3D program); discovery of proper two-phase localized, geometry-oblivious coloring algorithms for partitioning a WSN into backbones for applications of autonomous distributed requirements.

Chapter 3

GRAPHICAL PARTITIONING OF THE NATURAL NUMBER NETWORK

3.1. Introduction

The current confinement of numerical data to uninspiring chains of culturally determined digit choices has strangled the life out of numbers and arithmetic. Until recently, this situation was a result of the limitations of mechanical and early electronic devices. However, advances in inputting, processing, and data visualization, along with the emerging power of quantum computing, have now liberated us from these digital representation constraints.

The question now is, can we gain greater insight into “big data” from alternative forms of representation such as murals or 3D printed sculptures? Will this newfound freedom extend to arithmeticians, providing them with the inspirational capacity to realize operations in science that are equivalent to the experiences captured and preserved by artists in carvings and paintings across cultures over millennia?

To answer these questions, we introduce a graphical representation system of natural numbers (or counts) based on properties of prime numbers and the prime number function in Section 3.3. Each natural number is constructed by a rooted tree of Matula number [50] that can grow in two dimensions. Matula number describes a one-to-one mapping between natural numbers and rooted trees. This provides a countable alphabet of graphic symbols, with the size and form of each symbol reflecting both the size and the multiplicative structure of the number represented. Figure 3.1 shows the initial 21 coefficients of continued fraction of π rendered by this representation system. This culture-free representation system is founded on three fundamental principles of arithmetic, specifically a theorem (prime factorization), an operation (counting), and a computational process (recursion). The intent of a graphic system of natural numbers is to be concise, for both presentation and storage with precise

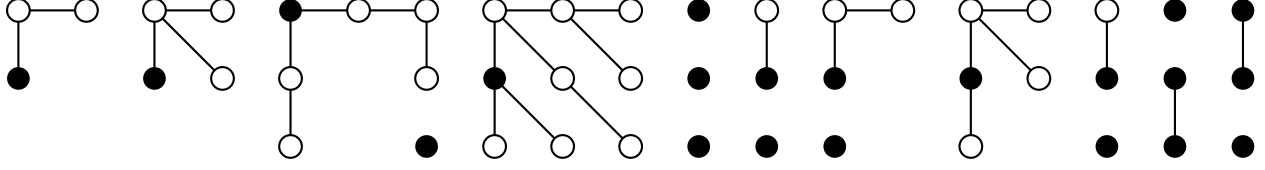


Figure 3.1: 21 Coefficients of Continued Fraction of π

completeness of range.

Furthermore, in Section 3.4, the adjacency relation between natural numbers is introduced by identifying the roots of all isomorphic Matula numbers belonging to the same automorphism group, drawing inspiration from the graphical representation of natural numbers. This also partitions the entire natural number network into finite sets, each corresponding to a non-isomorphic tree (termed “primordial tree”) within the forest of all finite trees. Properties and related theories will be illustrated in Section 3.5 with several 2D and 3D Visualized Algorithm Engineering (VAE) works. The potential for application of these graphics now unseen is hopefully suggested by a now well-known but previously unseen property of primes in the foundation of RSA cryptography.

3.2. Background

Primes are visually observed as those linear sequences of marks that cannot be reduced to a rectangular layout on a plane [65]. They have been discovered and themselves counted across cultures by those without schooling. In 1968, David W. Matula described a bijection between finite rooted trees (denoted by \mathbb{T}) and all natural numbers (positive integers, denoted by \mathbb{N}) [50]. The tree numbers are termed “Matula Number” with applications of encoding chemical structures [36]. Theorem 3.1 uses the notation p_k for the prime number of index k where $k \in \mathbb{N}$ to describe how the bijection is determined.

Theorem 3.1 *Matula number provides a one-to-one mapping $\mu(t) = n$ between $t \in \mathbb{T}$ and $n \in \mathbb{N}$ where:*

- (a) *The root of the single node rooted tree is labeled one ($n = 1$);*
- (b) *The directed edge out of each leaf is labeled with the prime $p_1 = 2$;*

- (c) An edge out of a node directed into its parent is labeled with the prime p_k where k is the product of the primes labeling the edges directed into the node;
- (d) For the root of any rooted tree having at least two nodes, the root label n is the product of all primes on the edges directed into the root.

Given a natural number $n \in \mathbb{N}$, the labeling can also be determined from root to leaves yielding the inverse mapping $t = \mu^{-1}(n)$ for $t \in \mathbb{T}$.

Proof:

$t \Rightarrow n$: Assuming the rooted tree has depth $j \geq 1$, all edges from depth j are labeled with $p_1 = 2$ as described in step (b). Then the labels on all edges from depth k to depth $k - 1$ ($1 < k < j$) are recursively assigned satisfying step (c). When all edges directed into the root have been labeled, then step (d) determines the root number n .

$n \Rightarrow t$: Determining $\mu^{-1}(n) \in \mathbb{T}$ for any natural number n , the edges from the root are labeled with the prime factors of n by unique prime factorization. For each prime factor p_k , the edge from the root leads to the subtree $\mu^{-1}(k)$. The procedure continues recursively as Algorithm 5 shows until $\mu^{-1}(1)$ denoting a single node (leaf) of the tree. \square

Algorithm 5 Construction of Rooted Tree of Matula Number

```

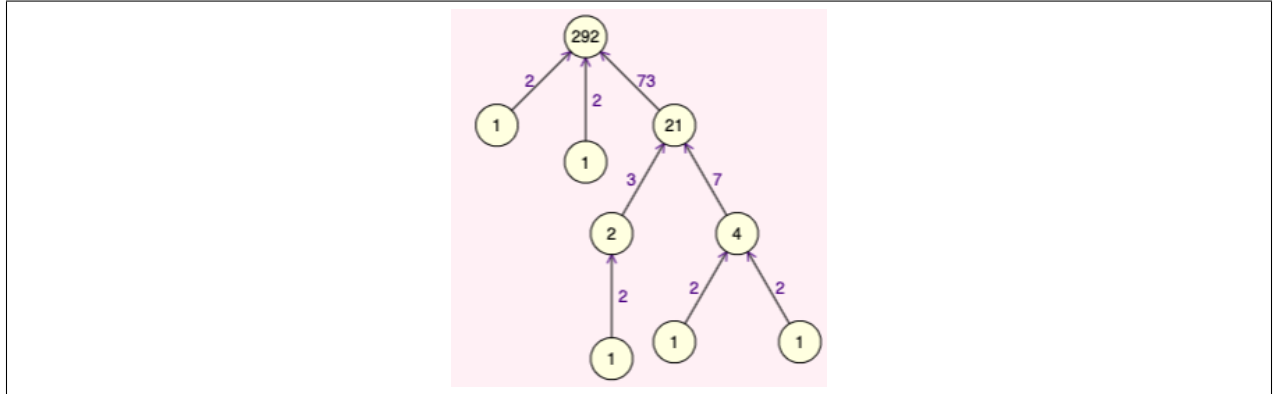
1: function MATULANUMBER(Number  $n$ )
2:    $T \leftarrow$  new Tree;
3:   BRANCH( $T, n$ );
4:   return  $T$ ;

5: procedure BRANCH(Tree  $T$ , Number  $n$ )
6:   Create a root node labeled  $n$ ;
7:   for each prime factor  $p_k$  of  $n$  do
8:     Create a branch edge labeled  $p_k$ ;
9:     BRANCH( $T, k$ );

```

An online animated Matula number generator has been created according to Algorithm 5 (URL: <https://s2.smu.edu/~zizhenc/Project/MatulaNumber>), which graphically presents the animation process of a Matula number growing from root to leaves. Screenshots of Matula numbers created by this program are used in the following sections and more

details of the implementation will be discussed in Section 3.5.2. Video 3.1 demonstrates the procedure of constructing the Matula number 292 rendered by the program, where the directed edges indicate the inverse procedure of how the root value 292 is calculated.



Video 3.1: [Matula Number 292](#)

3.3. Graphical Representation of Counts

3.3.1. Font Representation System

The Matula number provides us a precise graphical representation of each natural number as a rooted tree form factor. A grid representation is introduced by applying some regulations to the rooted trees of Matula number, which has the potential to be a number font system. Figure 3.2(a) shows a 3×3 grid symbol of Matula number 292. In comparison to the rooted tree of Matula number, the grid symbol undergoes a counterclockwise rotation of approximately 90 degrees (placing the root on the left) and distinguishes the root by coloring it black, in contrast to the other regulated white-colored nodes within a 3×3 grid. As a number font system, the tree structure of Matula number (shown in Figure 3.2(b)) can be treated as a natural form factor for human writing, while the grid representation can be treated as a mechanical form factor for print/display devices. Table 3.1 compares the grid symbols and the corresponding Matula numbers from 1 to 9. It is obvious that the number 3 and 4 have the same grid tree layout except different positions of the roots. This is not hard to explain by considering the isomorphic concept of graph theory. The number sets

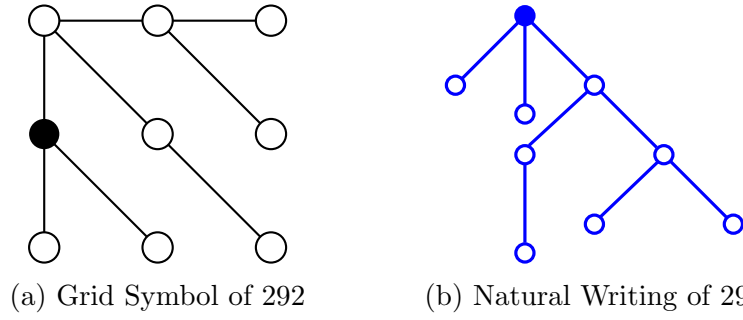










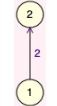
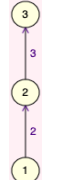
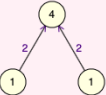
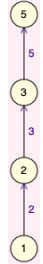
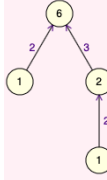
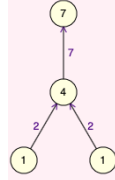
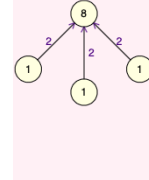
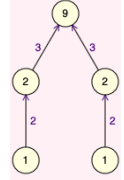





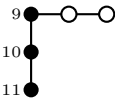
Figure 3.2: Graphic Representations of Natural Number 292

Table 3.1: Number 1~9 in Matula Number and Grid Graphic Representation

No.	1	2	3	4	5	6	7	8	9
Grid Rep.									
Matula No.									

$\{3, 4\}$, $\{5, 6\}$, $\{7, 8\}$ and $\{9, 10, 11\}$ belong to same automorphism groups of Matula numbers respectively. The numbers of each group can be rendered in a compact way by highlighting the roots (i.e., black-colored nodes) on the same grid tree layout as Table 3.2 shows. This

Table 3.2: Compact Grid Symbols for Numbers of Automorphism Groups*

Number Sets	$\{3, 4\}$	$\{5, 6\}$	$\{7, 8\}$	$\{9, 10, 11\}$
Grid Symbols				

allows the representation of 11 numbers (1 to 11) by only 6 grid symbols. Figure 3.3 compares the 11 natural numbers written in the Liquid-Crystal Display (LCD) font and in such grid representations. Furthermore, this compression method can also be applied to a series of

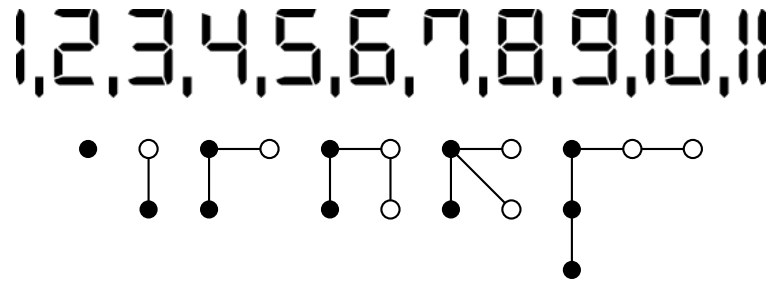
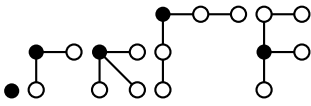










Figure 3.3: Comparison of LCD Font and Grid Representations of Numbers 1~11

repeating numbers. Among numbers from 2 to 13, as Table 3.3 shows, a series of repeating numbers can be rendered on the same grid tree layout by highlighting the roots on the symmetric branches. Obviously, not all natural numbers have such compact representation

(e.g.  ...). The number of possible repetition of each number on a same tree layout is also different and limited, which can be determined by the new tree concept (“primordial tree”) introduced in Section 3.4.

Unlike the traditional numeral system we use daily (e.g., Arabic numerals) that uses chains of culturally selected digits, this numbering scheme based on rooted trees has more

*The numbers marked besides the black nodes are not required in the compact grid representation. They help readers to understand the root position of the Matula number and its corresponding natural number.

Number Series	$\{2, 2\}$	$\{3, 3\}$	$\{5, 5\}$	$\{6, 6\}$	$\{7, 7, 7\}$	$\{10, 10\}$	$\{11, 11\}$	$\{13, 13\}$
Grid Symbols								

3,7,15,1,292,1,1,2,1,3,1,4,2,1,2,2,2,2,1

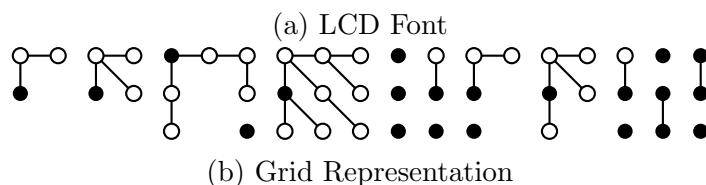


Figure 3.4: Initial 21 Coefficients of Continued Fraction of π^\dagger

large frequency of 1's and 2's in these coefficients is replicated in many expansions. For a random number picked uniformly over the interval $[0, 1)$, the frequency of small digits has been studied [40]. In particular, 1's occur about 41.5% of the time and 2's about 17.0%, both of whose representations fit in a single column. The frequency of partial coefficients in

†

[illegible]

the range one to eight is over 80% in total, and all of these integers are represented requiring no more than the 2×2 square grid. The occurrence of the particular partial coefficient 292 occurring in π 's list has in the random over $[0, 1)$ model a frequency of less than 0.002%. The partial coefficient 84 which is the next (22nd) in the continued fraction for π after those shown in Figure 3.4 has a frequency of only 0.02%. Note that a single column in our square grid representation is sufficient for over 58% of the partial coefficients, and another 22% fit in the 2×2 square grid.

3.3.2. Multiplicative Properties of the Graphical Representation of Counts

Section 3.3.1 introduces a 2D graphical font system based on the rooted tree of Matula number which is defined recursively according to prime factorization. This allows some fundamental arithmetic operations to be visually exploited. Given the prime factorization of two integers $u = p_1^{m_1} p_2^{m_2} \dots$ and $v = p_1^{m'_1} p_2^{m'_2} \dots$, the following multiplicative operations,

(a) **Multiplication:** $u \cdot v = p_1^{m_1 m'_1} p_2^{m_2 m'_2} \dots$

(b) **Division:** $u/v = n \in \mathbb{N}$, iff $m_1 \geq m'_1, m_2 \geq m'_2, \dots$

(c) **Greatest Common Divisor (GCD):** $GCD(u, v) = p_1^{\min\{m_1, m'_1\}} p_2^{\min\{m_2, m'_2\}} \dots$

(d) **Least Common Multiple (LCM):** $LCM(u, v) = p_1^{\max\{m_1, m'_1\}} p_2^{\max\{m_2, m'_2\}} \dots$

are immediate examples that can be deduced from the visual structure of the rooted trees. Definition 3.1 has been given here for further references.

Definition 3.1 A *prime branch* of a rooted tree of Matula number is a maximal subgraph containing a single child of the root. A prime branch is by itself a rooted tree where the root is then also a leaf.

By ordering the prime branches of the root of a Matula number, according to Theorem 3.1, some derivative observations of the multiplicative operations mentioned above can be visually made.

Observation 3.1 Given $n_1 = \mu(t_1)$, $n_2 = \mu(t_2)$ ($n_1, n_2 \in \mathbb{N}$ and $t_1, t_2 \in \mathbb{T}$), the integer product $n_1 \cdot n_2 = \mu(t_3)$ has $t_3 \in \mathbb{T}$ determined by merging all prime branches of t_1 and t_2 .

Observation 3.2 The determination of whether $\mu(t_1)$ exactly divides $\mu(t_2)$ ($t_1, t_2 \in \mathbb{T}$), and if so the resulting quotient $\mu(t_3)$, $t_3 \in \mathbb{T}$, is readily determined from the prime branches of t_1 and t_2 yielding the prime branches of t_3 .

Observation 3.3 For a set $\{t_1, t_2, \dots, t_i\} \in \mathbb{T}$, $i > 2$, the GCD and LCM of the set $\{\mu(t_1), \mu(t_2), \dots, \mu(t_i)\}$ is readily determined.

The three observations indicate that the results of some multiplicative operations can be simply “seen” from the structure of the rooted trees without calculations. Table 3.4 lists examples of multiplication, division, GCD and LCM conducted by the rooted trees of Matula number.

Table 3.4: Multiplicative Operations

(a) Multiplication and Division

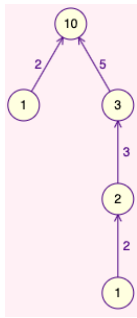
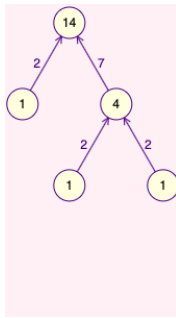
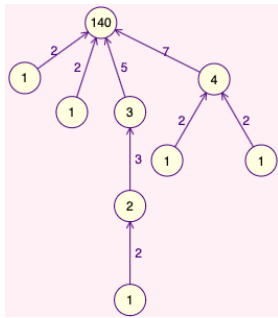
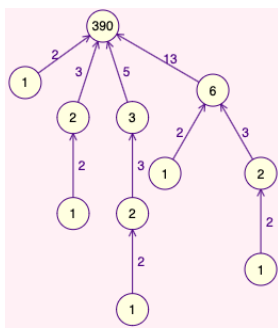
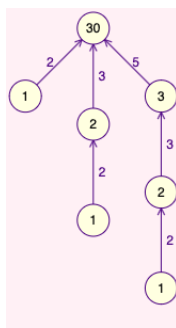
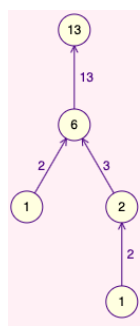
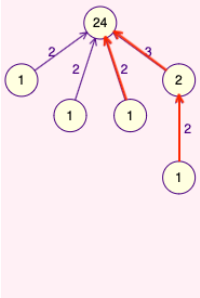
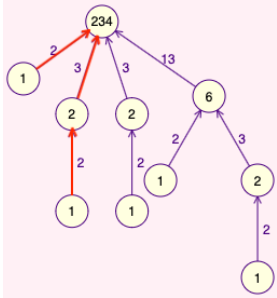
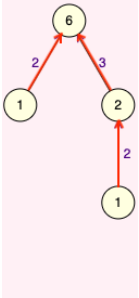
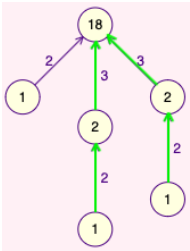
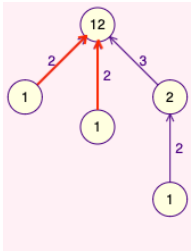
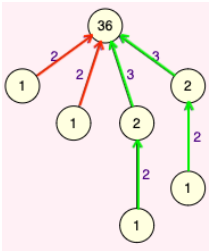
Operations	Rooted Trees of Matula Number				
10×14		merge		=	
$390 \div 30$		split		=	

Table 3.4: Multiplicative Operations, continued.

(b) GCD and LCM[‡]

Operations	Rooted Trees of Matula Number
$GCD(24, 234)$	$GCD($  ,  $) =$ 
$LCM(18, 12)$	$LCM($  ,  $) =$ 

3.3.3. Evaluations of the Graphical Number Representation

We introduced a graphical representation of natural numbers derived from arithmetic properties of primes. The definition of Matula number based on recursive computations provides a natural visualization of tree form factor for each natural number. The visual structure of each prime branch generated from the uniqueness of prime factorization has its own computational meaning in multiplicative calculations. This graphical numeral scheme has a real countable meaning behind it, because number zero can only be displayed as “nothing” or “empty”. That’s why we would rather call it “graphical representation of counts” than “graphical representation of natural numbers”[§]. Thus, this numeral representation is arguably fundamental as it comprises three fundamental principles of arithmetic:

- (a) A fundamental theorem — unique prime factorization is known as the fundamental theorem of arithmetic;
- (b) A fundamental operation — counting is a fundamental operation of arithmetic;
- (c) A fundamental process — recursion is a fundamental computational procedure.

More importantly, the visual forms for the counting numbers can be extended to a unique and concise numeric representation of the rationals by using coefficients of continued fractions (e.g., the 21-partial coefficients of continued fraction of π as Figure 3.4 shows).

A side product of this representation is the font representation system for natural numbers. The grid tree layout representation is suitable for print/display media, which has possible unseen potential applications. For example, for people who have visual impairments, the nodes of the rooted tree can be printed as mixed convex and concave dots (where convex dots represent the roots) with line slots as edges. The size of each grid symbol reveals

[‡]In the GCD operation, the red arrows highlight the common prime branches of number 24 and number 234, which is the result of the calculation. In the LCM operation, green arrows highlight the maximum number of prime branches of number 3, and red arrows highlight the maximum number of prime branches of number 2. The result is the rooted tree merging all highlighted prime branches.

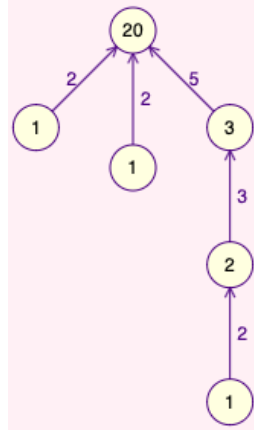
[§]This also eliminates the ambiguity caused by the definition of natural numbers (as positive integers or as non-negative integers [14])

how large the represented number is. The visual structure of this representation provides a favor to directly obtain multiplicative computational results.

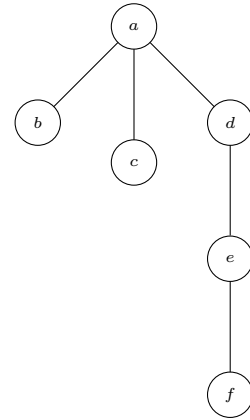
3.4. Adjacency Relation Between Natural Numbers

3.4.1. Non-isomorphic trees of Matula numbers

Considering the graph theoretic concept of the forest of all non-isomorphic trees (denoted by \mathbb{T}^*), a subset of natural numbers (denoted by \mathbb{N}^*) are naturally associated with each tree $t \in \mathbb{T}^*$. Here is an example that starts from the Matula number 20 as Figure 3.5 (a) shows. Figure 3.5 (b) provides the tree structure of this Matula number by excluding labels



(a) Matula Number 20

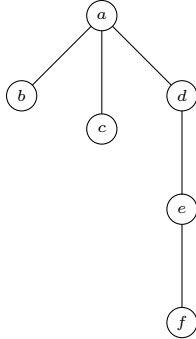
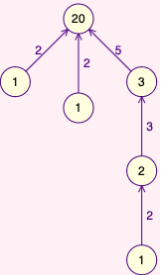
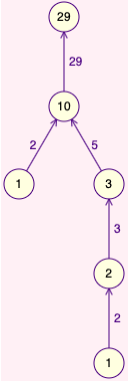
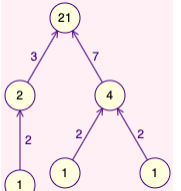
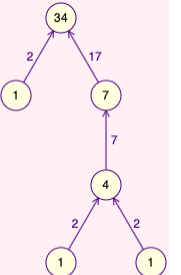
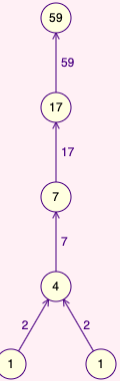


(b) The Structure of The Rooted Tree

Figure 3.5: The Structure of the Rooted Tree of Matula Number 20

and directions of edges, and replacing the labels of all nodes with alphabetic characters ($a \sim f$). Then each node of Figure 3.5 (b) can be used as a root to determine a rooted tree of Matula number t_i ($i = a, b, \dots, f$) by following the steps in Theorem 3.1 and each root is labeled $n_i = \mu(t_i)$. Obviously, for Matula number 20, the root is a and its label is $n_a = 20$. Table 3.5 lists all Matula numbers determined by all possible roots. Thus, the natural numbers in the set $\{20, 21, 29, 34, 59\}$ are associated by a directed graph t (shown in Figure 3.6) which is constructed based on the structure of Figure 3.5 (b) with the roots in Table 3.5 (a, b, \dots, f) as nodes. The directed edge information incident to each node is carried

Table 3.5: The Rooted Trees of Matula Number Set $\{20, 21, 29, 34, 59\}$

Root	a	$b c$	d	e	f
					

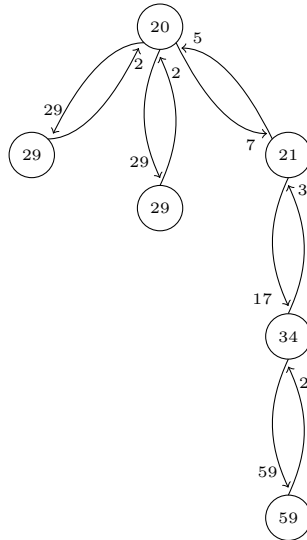


Figure 3.6: Non-isomorphic Tree of Number Set $\{20, 21, 29, 34, 59\}$

over from each corresponding associated rooted tree of Matula number. Such directed graph is a non-isomorphic tree termed “primordial tree” as Definition 3.2 states.

Definition 3.2 *A **primordial tree** is a non-isomorphic tree $t = T(V, E)$ representing all rooted trees of Matula number of a same automorphism group. V consists of all roots of the associated Matula numbers and E is composed of all directed edges incident to those roots in their corresponding Matula numbers.*

3.4.2. Specifications of Primordial Trees

According to Definition 3.2, the following two characteristics of a primordial tree can be directly derived.

- A primordial tree is an unrooted tree. There’s no special node (such as a root) to identify a non-isomorphic tree, but we can specify one as Definition 3.3 states.

Definition 3.3 *Each primordial tree $t_c \in \mathbb{T}^*$ may be uniquely identified with the **center node** which is the node (or nodes) with minimum label value c .*

- A primordial tree is a tree with double directed edge information since each node carries one edge directed into itself from its corresponding rooted tree of Matula number. Theorem 3.2 further describes the labeling of edges and nodes indicating the structure of a primordial tree.

Theorem 3.2 *Every primordial tree $t \in \mathbb{T}^*$ can have its nodes and edges (directed in both directions) labeled with the labels having the following two properties:*

- every directed edge from node u to node v is labeled with the prime corresponding to the prime branch from u into root node v ,*
- every node is labeled with the product of the primes on the edges directed into the node.*

Proof: The result can be verified by considering all prime branches determined for each root location in the primordial tree t with reference to Figure 3.6. \square

Each primordial tree associates a subset of natural numbers each of which is the label of one node of the tree. Considering all finite primordial trees, we have Theorem 3.3.

Theorem 3.3 *The set of all finite primordial trees \mathbb{T}^* partitions the natural number set \mathbb{N} into mutually exclusive subsets N^* of natural numbers.*

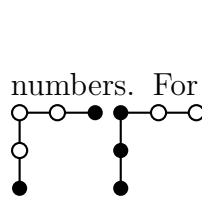
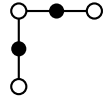
Proof: It is evident that all primordial tree set \mathbb{T}^* is a partition of the entire natural number network since each primordial tree associates a cluster of natural numbers. There's no overlap between any two clusters in this partition. Let N_1 and N_2 be two natural number sets ($N_1, N_2 \in N^*$) associated by two different primordial trees t_1 and t_2 ($t_1, t_2 \in \mathbb{T}^*$) respectively. Assume there is one natural number n belonging to both N_1 and N_2 ($n \in N_1 \wedge n \in N_2$), then there would exist one node labeled n in t_1 and one node also labeled n in t_2 . According to Theorem 3.1, there exists only one rooted tree $t = \mu^{-1}(n)$ of Matula number because of the uniqueness of prime factorization. Considering Definition 3.2, t_1 and t_2 can only be the same non-isomorphic tree, which contradicts the previous condition (t_1 and t_2 are two different primordial trees). Therefore, N_1 and N_2 are two mutually exclusive natural number sets. \square

The mapping from the associated number set N^* of a primordial tree $t = T(N, E) \in \mathbb{T}^*$ to the node set N is an injection function. In other words, there could be multiple nodes with a same label value in a primordial tree. Examples of this situation are $t_2, t_3, t_5, t_7, t_9, t_{12}...$ as listed in Table 3.6. Section 3.3.1 mentioned a compact graphic representation of series of repeating numbers by highlighting the root nodes on the symmetric branches of a single tree structure, which can be determined by such primordial trees. In fact, the idea of such compact representation (not only for series of repeating numbers) that utilizes alternative placement of the root node in a same tree layout complies with Definition 3.2, which makes the primordial tree become a natural generator of our graphical representation of natural

[¶]Nodes in red color are centrum nodes.

Table 3.6: Examples of Primordial Trees with Repeating Numbers[¶]

t_2	t_3	t_5	t_7	t_9	t_{12}


 numbers. For example, t_9 indicates that we can have compact grid representations
 
 for number series $\{10, 10\}$, $\{11, 11\}$ and $\{9, 10, 11\}$ respectively.

3.5. Visualized Algorithm Engineering

The preceding discussions have relied on the bijection mapping between natural numbers and rooted trees of Matula number. Theorem 3.1 outlines the recursive computational processes of this mapping which depends on the prime functions, namely, prime factorization, prime counting, and k_{th} prime. To facilitate most of the insights and observations, we employed the Matula number generator (URL: <https://s2.smu.edu/~zizhenc/Project/MatulaNumber>), an online graphic program designed by using the p5.js library. This section will delve into the fundamental algorithms of these prime functions. We will then expound on the implementation details of the Matula number generator program. Moreover, other VAE works will be explored, including primordial tree generator, primordial constellation, primordial garden, and so forth. These VAE works will also uncover some new discoveries. In the subsequent sections, the notation p_k will denote the k_{th} prime number, where $k \in \mathbb{N}$.

3.5.1. Algorithms of Prime Functions

3.5.1.1. Prime Factorization

Prime factorization is the theoretical foundation of constructing a Matula number from a given natural number. The straightforward idea of computing prime factors is to check the divisibility of n by all prime numbers less than or equal to itself. However, it is not necessary to determine primes among the natural numbers before checking divisibility. Instead we can keep dividing n by natural numbers in sequence[‡]. If the remaining quotient is divisible by a number, then this number must be a prime factor of n . The remaining quotient will never be divided by a composite number evenly since the quotient must be obtained by some former divisions of the prime factors of the composite number. Then we have a linear-time $O(n)$ performance algorithm of computing prime factors as Algorithm 6 shows. The performance

Algorithm 6 Prime Factorization

```
1: function PRIMEFACTOR(Number  $n$ )
2:    $L \leftarrow$  new List;
3:   for  $p \leftarrow 2$ ;  $n \geq p$ ;  $p \leftarrow p + 1$  do
4:      $n \leftarrow$  QUOTIENT( $L, n, p$ );
5:   return  $L$ ;

6: function QUOTIENT(List  $L$ , Number  $n$ , Number  $p$ )
7:   while  $n \bmod p = 0$  do
8:      $L.add(p)$ ;
9:      $n \leftarrow n/p$ ;
10:  return  $n$ ;
```

is not slow, but we still have two refinements to improve it.

- It is sufficient to check the divisibility of natural numbers starting from 2 to the number less than or equal to \sqrt{n} , as any larger number will become the remaining quotient if it can divide n evenly. Then the time complexity of the prime factorization has been improved to $O(\sqrt{n})$.

[‡]On hardware level, the more divisions that occur, the more time is consumed. Therefore, the second refinement mentioned below is also worthwhile.

- It is not necessary to check the divisibility of all natural numbers less than or equal to \sqrt{n} , as it takes trivial effort to filter out a lot of composite numbers among them. For example, all even numbers except 2 must be composite since they are all divisible by 2. In our implementation, a number sequence that filters out multiples of 2's or 3's is used, which is defined by Equation (3.1).

$$S_i = \begin{cases} 5 + 3 \cdot i, & \text{if } i \text{ is even} \\ 4 + 3 \cdot i, & \text{if } i \text{ is odd} \end{cases}, i \geq 0 \quad (3.1)$$

Yuri's paper [37] of prime-counting algorithm has more details of this number sequence.

Algorithm 7 shows the refined algorithm of prime factorization.

Algorithm 7 Refined Prime Factorization

```

1: function PRIMEFACTOR(Number  $n$ )
2:    $L \leftarrow$  new List;
3:    $n \leftarrow$  QUOTIENT( $L, n, 2$ );
4:    $n \leftarrow$  QUOTIENT( $L, n, 3$ );
5:    $i \leftarrow 0$ ;
6:    $p \leftarrow$  SEQUENCE( $i$ );
7:   while  $n \geq p^2$  do                                      $\triangleright \sqrt{n} \geq p$ 
8:      $n \leftarrow$  QUOTIENT( $L, n, p$ );
9:      $i \leftarrow i + 1$ ;
10:     $p \leftarrow$  SEQUENCE( $i$ );
11:  return  $L$ ;

12: function SEQUENCE(Number  $i$ )
13:  if  $i \bmod 2 = 0$  then
14:    return  $5 + 3 \times i$ ;
15:  else
16:    return  $4 + 3 \times i$ ;

```

3.5.1.2. Prime Counting and k_{th} Prime

The two following prime functions are frequently employed in our VAE works.

- k_{th} -prime: The k_{th} -prime function $p(k)$ returns the k_{th} prime number, $k \in \mathbb{N}$.

- Prime-counting: The prime-counting function calculates the quantity of prime numbers that are less than or equal to a given real number r [4]. In our specific use cases, r is always a prime number, which makes the prime-counting function the inverse of the k_{th} -prime function $p^{-1}(r)$.

Both prime functions are, in fact, concerned with the same problem of determining prime numbers among natural numbers. The sieve of Eratosthenes, an ancient algorithm for finding all prime numbers up to a given limit [38], can be used to solve this problem. It works by iteratively removing the multiples of each prime (since they are composite), starting with 2 as Algorithm 8 shows. Figure 3.7 shows the procedure of sieving numbers up to 20 and the

Algorithm 8 The Sieve of Eratosthenes

```

1: function SIEVE(Number  $n$ )
2:    $L \leftarrow \{2, 3, \dots, n\}$ ;
3:    $P \leftarrow$  new List;
4:   while  $L$  is not empty do
5:      $p \leftarrow L.\text{popFirst}()$ ; ▷ return and remove first element of the list
6:      $P.\text{add}(p)$ ;
7:      $i \leftarrow 2$ ;
8:     while  $i \times p \leq L.\text{getLast}()$  do
9:        $L.\text{remove}(i \times p)$ ;
10:     $i \leftarrow i + 1$ 
11:  return  $P$ ;

```

resulting primes are $\{2, 3, 5, 7, 13, 17\}$. After the multiples of 2 and 3 have been removed,

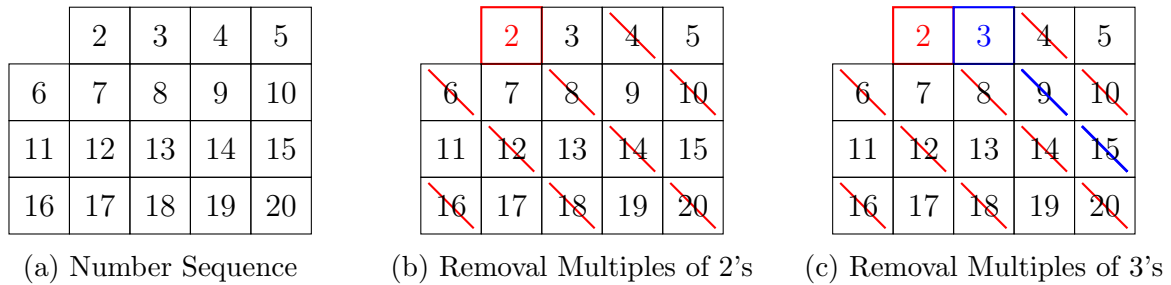


Figure 3.7: The Sieve of Eratosthenes Applied on Numbers Up to 20

the algorithm actually terminates since the multiples of 5 and 7 have already been removed by former operations, and multiples of 13, 17 and 19 are larger than $n = 20$. Therefore we have the following two refinements of Algorithm 8.

- It is sufficient to traverse the numbers starting from the square of each prime (p^2) instead of 2 (Line 7: $i \leftarrow p$), as all the smaller multiples of p will have already been removed at that point. Then the algorithm is allowed to terminate when $p^2 > n$.
- Similar to Algorithm 7, instead of traversing all numbers less than or equal to n , we can traverse numbers of the sequence of Equation (3.1) along with 2 and 3 (Line 2: $L \leftarrow \{i | i = 2, 3, \text{sequence of Equation (3.1), } i \leq n\}$).

The time complexity of this algorithm lies in the times of removing operations. The first iteration removes all multiples of first prime number 2, which has about $\frac{n}{2}$ number of times; the second iteration removes multiples of the 2_{nd} prime number 3, which has about $\frac{n}{3}$ number of times; then the third iterations has about $\frac{n}{5}$ numbers of times;... and so forth. The total number times would be

$$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \dots = n \times \sum_{p_k \leq \sqrt{n}} \frac{1}{p_k} \geq \log \log (n+1) - \log \frac{\pi^2}{6}, \quad k \in \mathbb{N}$$

[25]. So the time complexity is $O(n \cdot \log \log n)$.

To solve the two prime functions, an array of marks can be created that assigns mark 1's to the prime number locations and 0's to the composite number locations. The prime-counting function then calculates the sum of marks before the given number, while the k_{th} prime function returns the prime number corresponding to the k_{th} 1-mark in the sequence as shown in Algorithm 9. The time complexity of the two prime functions relies on the time complexity of the sieve of Eratosthenes algorithm which is $O(n \cdot \log \log n)$.

3.5.2. Matula Number Generator

The data structure used for the Matula number generator program to store the rooted trees is an adjacency list with edge information as Figure 3.8 shows. Although theoretically,

^{**}Several studies have been conducted to calculate the upper bound of p_k [3, 22, 42], and the formulas resulting from these researches will only impact the algorithm's space complexity. In the practical implementation, the array of marks and the list of primes can be declared as global variables, thereby saving the redundant time complexity for invoking prime functions multiple times.

Algorithm 9 Prime-counting and k_{th} -prime Functions

```

1: function PRIMECOUNTING(Number  $n$ )
2:    $P \leftarrow \text{MARK}(n)$ ;
3:   return  $P[n]$ ;

4: function  $k_{th}$ PRIME(Number  $k$ )
5:    $n \leftarrow$  some upper bound of  $p_k$ ;**
6:    $N \leftarrow \text{MARK}(n)$ ;
7:   return  $N[k]$ ;

8: function MARK(Number  $n$ )
9:    $M \leftarrow$  new Array[ $n + 1$ ];            $\triangleright$  All values are set to 0, index starts from 0
10:   $P \leftarrow \text{SEIVE}(n)$ ;
11:  for each  $p$  of  $P$  do
12:     $M[p] \leftarrow 1$ ;
13:  for  $i \leftarrow 1; i \leq n; i \leftarrow i + 1$  do
14:     $M[i] \leftarrow M[i - 1] + M[i]$ ;        $\triangleright M[i]$  stores the number of primes before number  $i$ 
15:  return  $M$ ;

```

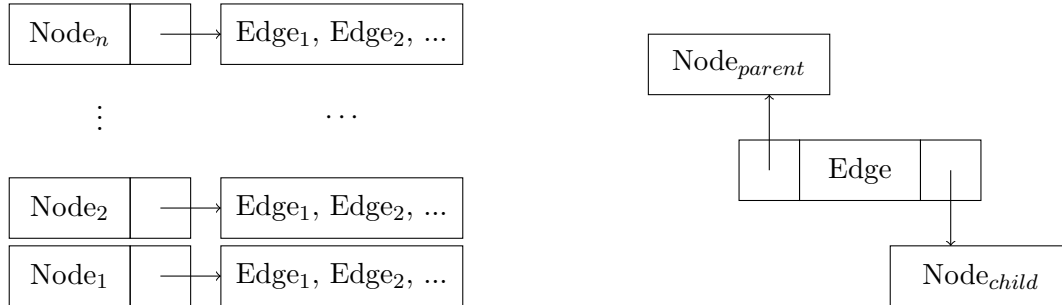


Figure 3.8: Adjacency List with Edges

a classic adjacency list without edges is capable to describe a tree, considering visualization purposes, it would be much easier to use a data structure with dedicated edge information to draw directed edges. In this structure, each node has a list of edges each connecting a parent node and a child node. Additionally, each edge also needs to store the specific prime factor of the parent node for the connected child node, which is termed “stem factor” to indicate the direction from child to parent.

Algorithm 5 has provided the main recursive computational procedure of constructing a rooted tree for a given natural number. Both the prime factorization (to get p_k in Line 7) and the prime-counting function (to calculate k in Line 9) are necessary, which can be implemented according to Algorithm 7 and Algorithm 9 discussed in Section 3.5.1. One additional adjustment of Algorithm 5 that needs to be made is to convert the recursive procedure to an iterative procedure through a stack data structure. Since our VAE work requires the creation of an animation that demonstrates the generation of a Matula number, an algorithm of iterative procedure is necessary to display each state of the rooted tree during the construction process. Algorithm 10 provides the main construction algorithm for the Matula number generator and it returns the root of the created tree which can be the input of either Depth First Search (DFS) or Breadth First Search (BFS) algorithm to display the rooted tree in linear time.

Algorithm 10 Matula Number Generator

```

1: function MATULANUMBER(Number  $n$ )
2:    $S \leftarrow$  new Stack;
3:    $S.push(\text{new Node}(n));$                                  $\triangleright$  Create a new node as root labeled  $n$ 
4:   while  $S$  is not empty do
5:      $\text{node} \leftarrow S.pop();$ 
6:      $F \leftarrow \text{PRIMEFACTOR}(\text{node.label});$ 
7:     for each  $p$  of  $F$  do
8:        $k \leftarrow \text{PRIMECOUNTING}(p);$ 
9:        $\text{child} \leftarrow \text{new Node}(k);$ 
10:       $\text{edge} \leftarrow \text{new Edge}(\text{node}, \text{child});$        $\triangleright$  add edge to edge lists of node and child
11:       $\text{edge.stemFactor} \leftarrow p;$ 
12:       $S.push(\text{child});$ 
13:   return root;

```

The time complexity of Algorithm 10 relies on the time used on prime factorization and prime counting. For the prime counting part, only the first function call on the root label value takes $O(n \cdot \log \log n)$ time, the subsequent recursive calls take only $O(1)$ constant time since their results can be retrieved from the array generated by the initial call. For the prime factorization part, each node of the Matula number needs to call the prime factorization function once. Studies on Gutman-Ivić-Matula function [10, 23] provide the upper bound of the size (number of edges) of a Matula number, which is $\frac{3}{\log 5} \cdot \log n$, then the upper bound of the order (number of nodes) is $\frac{3}{\log 5} \cdot \log n + 1$. Therefore, the time complexity of prime factorization part of Algorithm 10 can be estimated as $O(\sqrt{n} \cdot \log n)^{\dagger\dagger}$ which is less than $O(n \cdot \log \log n)$ when n is sufficiently large. After all, the time complexity of Algorithm 10 is still $O(n \cdot \log \log n)$.

3.5.3. Primordial Tree Generator

The Primordial Tree Generator is a web-based program that generates a primordial tree from a given natural number, which can be accessed at <https://s2.smu.edu/~zizhenc/Project/PrimordialTree>. In the upcoming sections, various implementation methods will be presented along with observations and visualizations originated from the primordial tree generator.

3.5.3.1. Brute Force Method of Primordial Tree Construction

From Section 3.4.2, we learned that a primordial tree is an unrooted tree with double directed edges. To visualize the primordial tree, we can still use the data structure shown in Figure 3.8. The “parent” and “child” concepts in the structure are suitable to describe the visual positions of the nodes, as an unrooted tree still requires a visual “root” node to be presented as a tree form factor. The visual “root” node is selected to be the node labeled the natural number of the input. One slight adjustment that needs to be made to

^{††}This is a rough estimation because prime factorization may take different amounts of time for nodes of varying heights in the tree. The leaves require no time, and the children of the root node may take significantly less time than the root node itself.

the adjacency list is that the prime factor information of each edge should consist of two factors, each representing one direction. The stem factor means the specific prime factor of the parent node for the edge, which has the direction from the child node to the parent node. Correspondingly, the prime factor for the other direction is termed “branch factor”.

According to Definition 3.2, the straightforward method to create a primordial tree involves three procedures:

- (a) Create a rooted tree of Matula number from a given natural number.
- (b) Determine labels of all the remaining nodes each maps a Matula number of the same tree structure.
- (c) Carry over the directed edge information of all root nodes to construct the double directed edges.

Procedure (a) can be effectively solved by Algorithm 10. For the procedure (b), Theorem 3.1 describes the steps for bottom-to-top calculation of the root label value from leaves. Using the DFS algorithm to traverse the nodes of the tree, we can derive Algorithm 11 to determine the root label value of a given tree structure of Matula number. Note that Algorithm 11 makes use of the marks of nodes to traverse from the root to leaves and the marks of edges to backtrack from leaves to root. Meanwhile, recording the label values (at Line 3) of all root nodes and the directions of their incident edges, as well as the corresponding prime factors (at Line 19), is necessary to successfully complete the procedure (c). In this way, a primordial tree can be constructed from a given natural number as an input. Video 3.2 demonstrates the constructing procedure of the tree t_{147} that originates from node labeled 292. The centrum node is labeled 147 and highlighted in red color.

3.5.3.2. Integer Connectivity

Theorem 3.1 reveals that for any rooted tree of Matula number, each edge directed out of a node may be labeled by a prime p_k with a product k readily determined from the appropriate labeled edges directed into that node. As a primordial tree connects a cluster

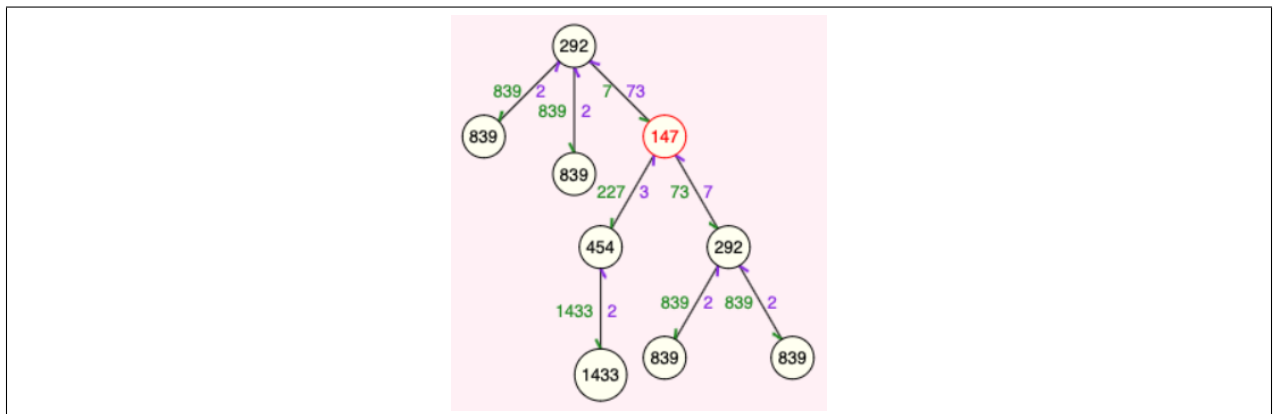
Algorithm 11 Root Label Determination

```

1: function ROOTVALUE(Node root)
2:   Clean marks of nodes and edges
3:   return TRAVERSE(root);

4: function TRAVERSE(Node node)
5:   node.mark  $\leftarrow$  true; ▷ mark node as traversed
6:   if node is a leaf then
7:     return COMPUTEVALUE(node, 1);
8:   else
9:     product  $\leftarrow$  1;
10:    for each edge of node.edges do
11:      neighbor  $\leftarrow$  connected node of edge;
12:      if neighbor.mark=false then ▷ if neighbor hasn't been traversed
13:        product  $\leftarrow$  product  $\times$  TRAVERSE(neighbor);
14:    return COMPUTEVALUE(node, product);

15: function COMPUTEVALUE(Node node, Number value)
16:   if node is root then ▷ if node is root, return label value
17:     return value;
18:   else ▷ otherwise, return prime factor of the parent
19:     edge  $\leftarrow$  unmarked edge of node; ▷ only one untraversed edge connecting parent
20:     edge.mark  $\leftarrow$  true; ▷ mark edge as traversed
21:     return  $k_{th}$ PRIME(value);
  
```



Video 3.2: [Primordial Tree \$t_{147}\$](#)

of natural numbers each of which corresponds to a rooted tree of Matula number, with the aid of the primordial tree generator online program, the following observation has been discovered.

Observation 3.4 *For any pair of two positive integers i and j ($i, j \in \mathbb{N}$), there must exist an adjacency relation between $i \cdot p_j$ and $j \cdot p_i$ ($i \cdot p_j \mathbb{R} j \cdot p_i$) established by a primordial tree containing two neighbor nodes labeled $i \cdot p_j$ and $j \cdot p_i$ respectively.*

This can be more easily verified by rewriting the prime factors as the k_{th} prime function $p(k)$ form factor on the directed edges of a primordial tree, as demonstrated in Figure 3.9.

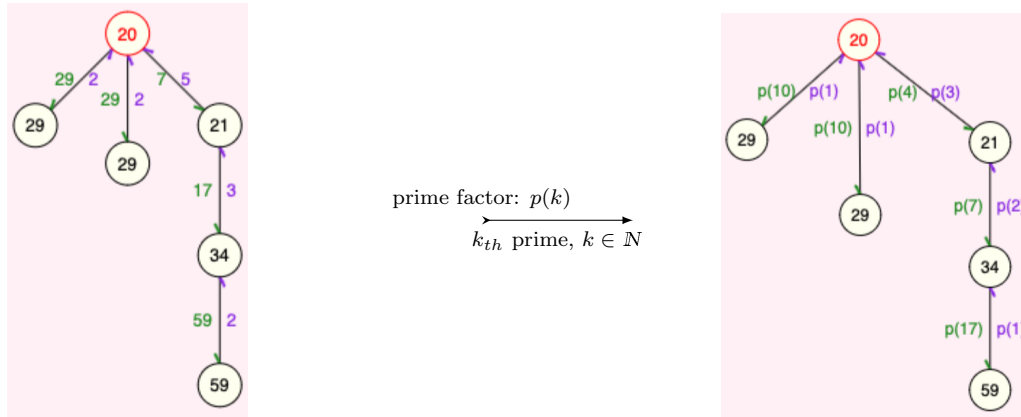


Figure 3.9: Primordial Tree t_{20}

It is not difficult to show the reflexive property: $i \cdot p_i \mathbb{R} i \cdot p_i$ and the symmetric property: $i \cdot p_j \mathbb{R} j \cdot p_i \Leftrightarrow j \cdot p_i \mathbb{R} i \cdot p_j$. This adjacency relation \mathbb{R} can readily be extended to an equivalence relation termed “Integer Connectivity”. The resulting equivalence relation partition of \mathbb{N} under this integer connectivity relation is readily shown to be identical to the tree set integer partition.

3.5.3.3. Primordial Tree Construction via Integer Connectivity

The method for constructing a primordial tree, as described in Section 3.5.3.1, involves a significant number of redundant computations and is essentially a brute force method that is derived from the formal definition of a primordial tree (as stated in Definition 3.2). Section 3.5.3.2 introduced the concept of integer connectivity ($i \cdot p_j \mathbb{R} j \cdot p_i$, $i, j \in \mathbb{N}$), and this

led to the development of a more efficient method (Algorithm 12) based on this relation. The

Algorithm 12 Primordial Tree Generator

```

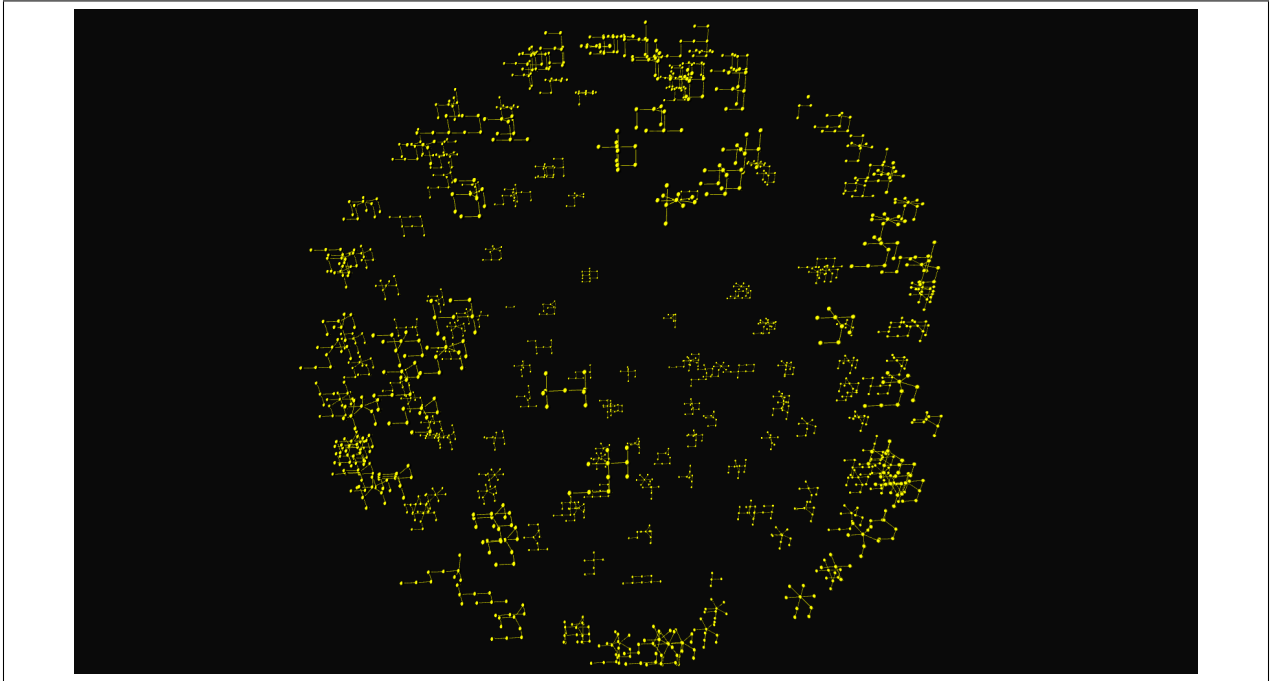
1: function PRIMORDIALTREE(Number  $n$ )
2:   node  $\leftarrow$  new Node( $n$ );
3:   return node;

4: procedure BRANCH(Node node)
5:   if node is the visual root then
6:      $F \leftarrow$  PrimeFactor(node.label);
7:   else
8:      $e \leftarrow$  node.edges.getFirst();  $\triangleright$  node only have one edge.
9:      $F \leftarrow$  PrimeFactor(node.label/e.branchFactor);
10:  for each  $p_i$  of  $F$  do
11:     $j \leftarrow$  node.label/ $p_i$ ;  $\triangleright$  node.label =  $j \cdot p_i$ 
12:    child  $\leftarrow$  new Node(PRIMECOUNTING( $p_i$ )  $\times$   $k_{th}$ PRIME( $j$ ));  $\triangleright$  child.label =  $i \cdot p_j$ 
13:    edge  $\leftarrow$  new Edge(node, child);  $\triangleright$  add edge to edge lists of node and child
14:    edge.stemFactor  $\leftarrow$   $p_i$ ;
15:    edge.branchFactor  $\leftarrow$   $p_j$ ;
16:    BRANCH(child);

```

current version of the primordial tree generator online program is built on Algorithm 12, but with modification to convert the recursive procedure into iterative procedure for visualization purposes, similar to what was done for the Matula number generator program discussed in Section 3.5.2.

This algorithm could also be the basis for further VAE works, such as creating an animation to demonstrate how the entire network of natural numbers is partitioned. A web-based animation program called “Primordial Constellation” has been implemented (URL: <https://s2.smu.edu/~zizhenc/Project/PrimordialConstellation>), which randomly distributes some amount of primordial trees on the surface of a sphere. The program accepts a number as the input to determine the amount of trees to be distributed and each tree is identified and originated by a centrum node. Video 3.3 showcases the process of constructing 200 primordial trees that are randomly distributed on a sphere, which is rendered by the program. The ability to zoom in and explore the trees from the inside out, as shown in Figure 3.10, is the reason why this visualization is named “Primordial Constellation”.



Video 3.3: [Primordial Constellation](#)

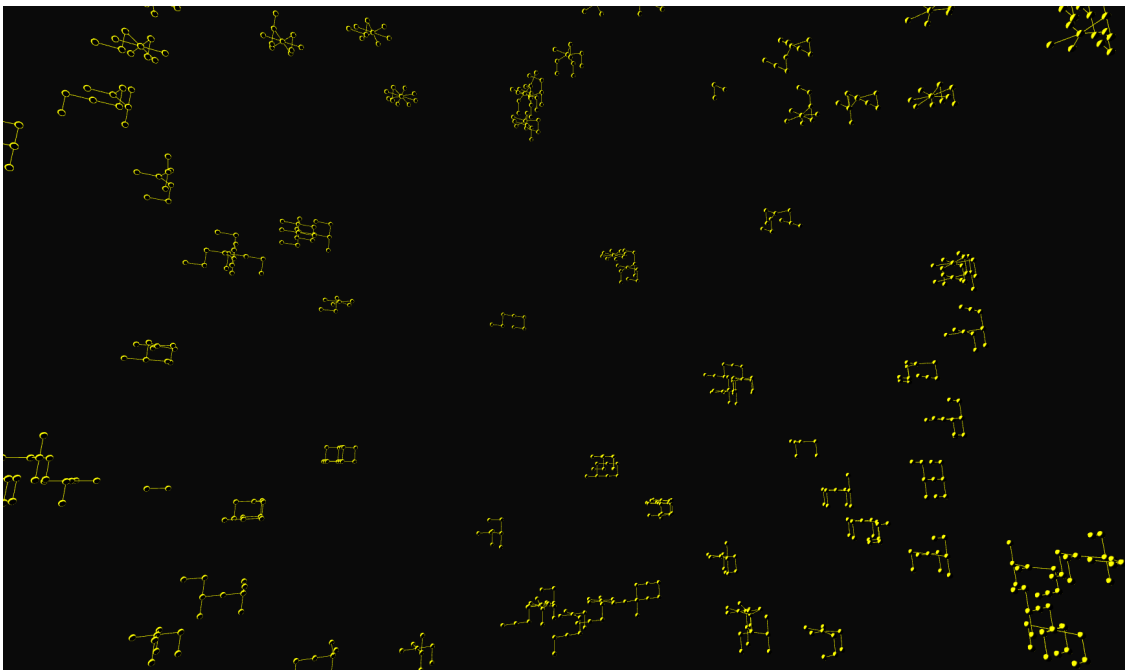


Figure 3.10: Observe the Exterior from within the Primordial Constellation

3.5.4. $i \cdot p_j$ Matrix

According to Observation 3.4, a primordial tree not only groups a set of natural numbers but also delineates the relationships between $i \cdot p_j$, $i, j \in \mathbb{N}$. This observation motivated us to devise a matrix data structure defined by Definition 3.4.

Definition 3.4 An $i \cdot p_j$ **matrix** S_m is composed of the first m multiples of the first m primes, with the entry at cell (j, i) containing the value $i \cdot p_j$.

Figure 3.11 displays S_{20} , which was generated by a web-based program: <https://s2.smu.edu/~zizhenc/Project/iPjMatrix/>. In growing the matrix from S_m to S_{m+1} , the column

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P(1)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
P(2)	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
P(3)	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P(4)	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
P(5)	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
P(6)	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
P(7)	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
P(8)	19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
P(9)	23	46	69	92	115	138	161	184	207	230	253	276	299	322	345	368	391	414	437	460
P(10)	29	58	87	116	145	174	203	232	261	290	319	348	377	406	435	464	493	522	551	580
P(11)	31	62	93	124	155	186	217	248	279	310	341	372	403	434	465	496	527	558	589	620
P(12)	37	74	111	148	185	222	259	296	333	370	407	444	481	518	555	592	629	666	703	740
P(13)	41	82	123	164	205	246	287	328	369	410	451	492	533	574	615	656	697	738	779	820
P(14)	43	86	129	172	215	258	301	344	387	430	473	516	559	602	645	688	731	774	817	860
P(15)	47	94	141	188	235	282	329	376	423	470	517	564	611	658	705	752	799	846	893	940
P(16)	53	106	159	212	265	318	371	424	477	530	583	636	689	742	795	848	901	954	1007	1060
P(17)	59	118	177	236	295	354	413	472	531	590	649	708	767	826	885	944	1003	1062	1121	1180
P(18)	61	122	183	244	305	366	427	488	549	610	671	732	793	854	915	976	1037	1098	1159	1220
P(19)	67	134	201	268	335	402	469	536	603	670	737	804	871	938	1005	1072	1139	1206	1273	1340
P(20)	71	142	213	284	355	426	497	568	639	710	781	852	923	994	1065	1136	1207	1278	1349	1420

Figure 3.11: 20×20 $i \cdot p_j$ Matrix S_{20}

being added contains the cells each having the value $(m+1) \cdot p_k$, where $1 \leq k \leq m$ and the row being added is the multiples of the new prime p_{m+1} up to $(m+1) \cdot p_{m+1}$. The following characteristics of $i \cdot p_j$ matrix are relevant to primordial trees.

- (a) Each natural number occurs a number of times equal to its count of distinct prime factors. For instance, in S_{20} , the number 30 appears at $(3, 6)$, $(2, 10)$ and $(1, 15)$ locations,

corresponding to its factorizations of $6 \cdot p_3$, $10 \cdot p_2$ and $15 \cdot p_1$, respectively.

- (b) Clearly, any pair of cells in the $i \cdot p_j$ matrix that are symmetric to the main diagonal represent neighboring nodes in a primordial tree because of the relation $i \cdot p_j \mathbb{R} j \cdot p_i$.
- (c) Figure 3.11 shows that centrum nodes of primordial trees are more likely to be labeled with values in the upper triangular area of the $i \cdot p_j$ matrix, as highlighted in purple. The reason behind this is that the value of k_{th} prime function $p(k)$ increases much faster than the linear value k .
- (d) In the relation $i \cdot p_j \mathbb{R} j \cdot p_i$, if $i \cdot p_j = j \cdot p_i$, then there are two adjacent nodes with the same label value $i \cdot p_j$ (or $j \cdot p_i$) in a primordial tree. The $i \cdot p_j$ matrix exhibits this property, where all cells on the main diagonal contain values each labeling two adjacent nodes in primordial trees since $i \cdot p_i = i \cdot p_i$. Furthermore, there are cells outside the main diagonal that contain the same values as those on the main diagonal, due to the characteristic mentioned in the bullet point (a). Figure 3.12 shows all cells that meet $i \cdot p_j = j \cdot p_i$ in S_{20} .

Because of these characteristics, with a sufficient large $i \cdot p_j$ matrix, all numbers of the group clustered by a primordial tree can be identified. Given a natural number n as input, all nodes (cells) with the same label value of n can be identified from the $i \cdot p_j$ matrix. From there, the neighboring nodes can be determined by examining the cells that are symmetric to the main diagonal. Recursively searching for cells with the same values as the neighbors, all other numbers connected by a same primordial tree can be discovered. Figure 3.13 displays the popped-up cells corresponding to the numbers clustered by the primordial tree t_{20} (refers to Figure 3.9).

3.5.5. Primordial Spiral

The prime spiral (also known as Ulam's spiral) is a graphical depiction of the set of prime numbers, devised by mathematician Stanislaw Ulam in 1963, who noticed that the prime numbers tended to cluster along certain curves or spirals [29]. The prime numbers are placed

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P(1)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
P(2)	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
P(3)	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P(4)	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
P(5)	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
P(6)	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
P(7)	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
P(8)	19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
P(9)	23	46	69	92	115	138	161	184	207	230	253	276	299	322	345	368	391	414	437	460
P(10)	29	58	87	116	145	174	203	232	261	290	319	348	377	406	435	464	493	522	551	580
P(11)	31	62	93	124	155	186	217	248	279	310	341	372	403	434	465	496	527	558	589	620
P(12)	37	74	111	148	185	222	259	296	333	370	407	444	481	518	555	592	629	666	703	740
P(13)	41	82	123	164	205	246	287	328	369	410	451	492	533	574	615	656	697	738	779	820
P(14)	43	86	129	172	215	258	301	344	387	430	473	516	559	602	645	688	731	774	817	860
P(15)	47	94	141	188	235	282	329	376	423	470	517	564	611	658	705	752	799	846	893	940
P(16)	53	106	159	212	265	318	371	424	477	530	583	636	689	742	795	848	901	954	1007	1060
P(17)	59	118	177	236	295	354	413	472	531	590	649	708	767	826	885	944	1003	1062	1121	1180
P(18)	61	122	183	244	305	366	427	488	549	610	671	732	793	854	915	976	1037	1098	1159	1220
P(19)	67	134	201	268	335	402	469	536	603	670	737	804	871	938	1005	1072	1139	1206	1273	1340
P(20)	71	142	213	284	355	426	497	568	639	710	781	852	923	994	1065	1136	1207	1278	1349	1420

Figure 3.12: $i \cdot p_j = j \cdot p_i$ Cells of S_{20}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P(1)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
P(2)	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
P(3)	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
P(4)	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
P(5)	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
P(6)	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
P(7)	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
P(8)	19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
P(9)	23	46	69	92	115	138	161	184	207	230	253	276	299	322	345	368	391	414	437	460
P(10)	29	58	87	116	145	174	203	232	261	290	319	348	377	406	435	464	493	522	551	580
P(11)	31	62	93	124	155	186	217	248	279	310	341	372	403	434	465	496	527	558	589	620
P(12)	37	74	111	148	185	222	259	296	333	370	407	444	481	518	555	592	629	666	703	740
P(13)	41	82	123	164	205	246	287	328	369	410	451	492	533	574	615	656	697	738	779	820
P(14)	43	86	129	172	215	258	301	344	387	430	473	516	559	602	645	688	731	774	817	860
P(15)	47	94	141	188	235	282	329	376	423	470	517	564	611	658	705	752	799	846	893	940
P(16)	53	106	159	212	265	318	371	424	477	530	583	636	689	742	795	848	901	954	1007	1060
P(17)	59	118	177	236	295	354	413	472	531	590	649	708	767	826	885	944	1003	1062	1121	1180
P(18)	61	122	183	244	305	366	427	488	549	610	671	732	793	854	915	976	1037	1098	1159	1220
P(19)	67	134	201	268	335	402	469	536	603	670	737	804	871	938	1005	1072	1139	1206	1273	1340
P(20)	71	142	213	284	355	426	497	568	639	710	781	852	923	994	1065	1136	1207	1278	1349	1420

Figure 3.13: Nodes of Primordial Tree t_{20}

on the spiral that they form a distinctive pattern that is both beautiful and mysterious as Figure 3.14 illustrates. Building upon this inspiration, we developed a web-based program

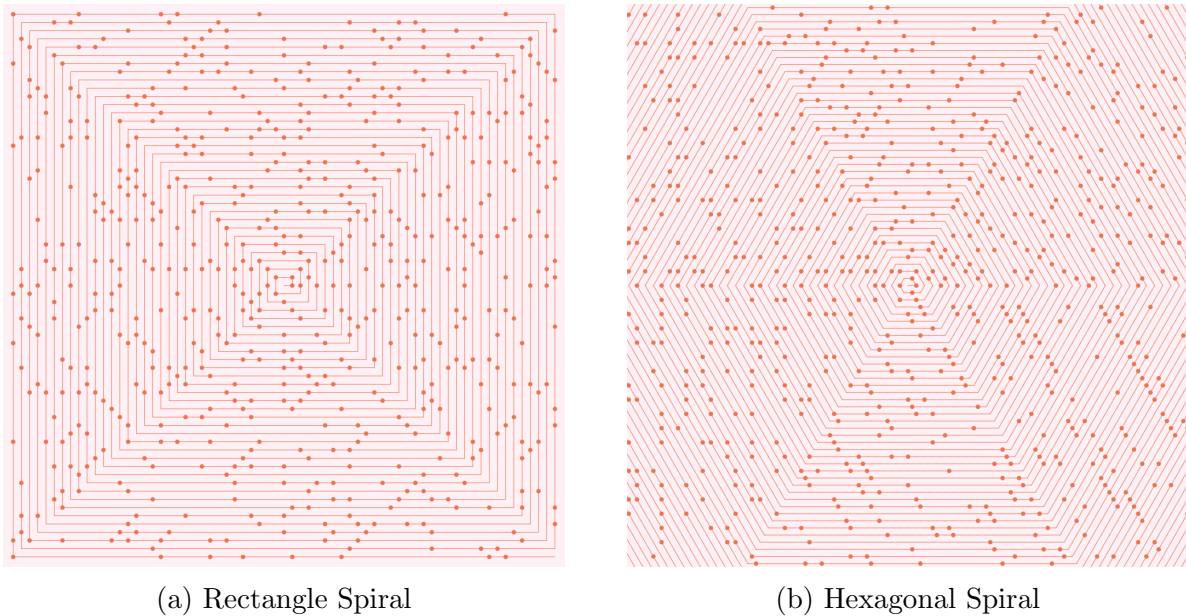


Figure 3.14: Prime Spiral

that displays the number spiral as shown in Figure 3.15, which can be accessed at <https://s2.smu.edu/~zizhenc/Project/PrimordialSpiralPattern>. The program features a color scheme that highlights prime numbers in orange, composite numbers in gray, and encircles the numbers labeling centrum nodes (termed “centrum number”) of primordial trees. It can also display a hexagonal number spiral similar to Figure 3.14 (b).

Note that the centrum numbers also cluster along certain curves as illustrated by Figure 3.16 which shows the centrum numbers (as circles) only and we term this spiral as “Primordial Spiral”. The spiral has yielded yet another discovery — only the centrum numbers 2, 3, 5, and 7 are prime. All other centrum numbers are composite, as clearly illustrated in Figure 3.17 where displays the prime and centrum numbers only.

Proof: First, Figure 3.18 lists the primordial trees t_2, t_3, t_5, t_7 , which shows that the prime numbers 2, 3, 5, and 7 are centrum numbers. According to the $i \cdot p_j$ matrix (illustrated in Figure 3.11), any node labeled with a prime number greater than 7 must have an adjacent node with a label value greater than that node. In Section 3.5.4, it was described that any

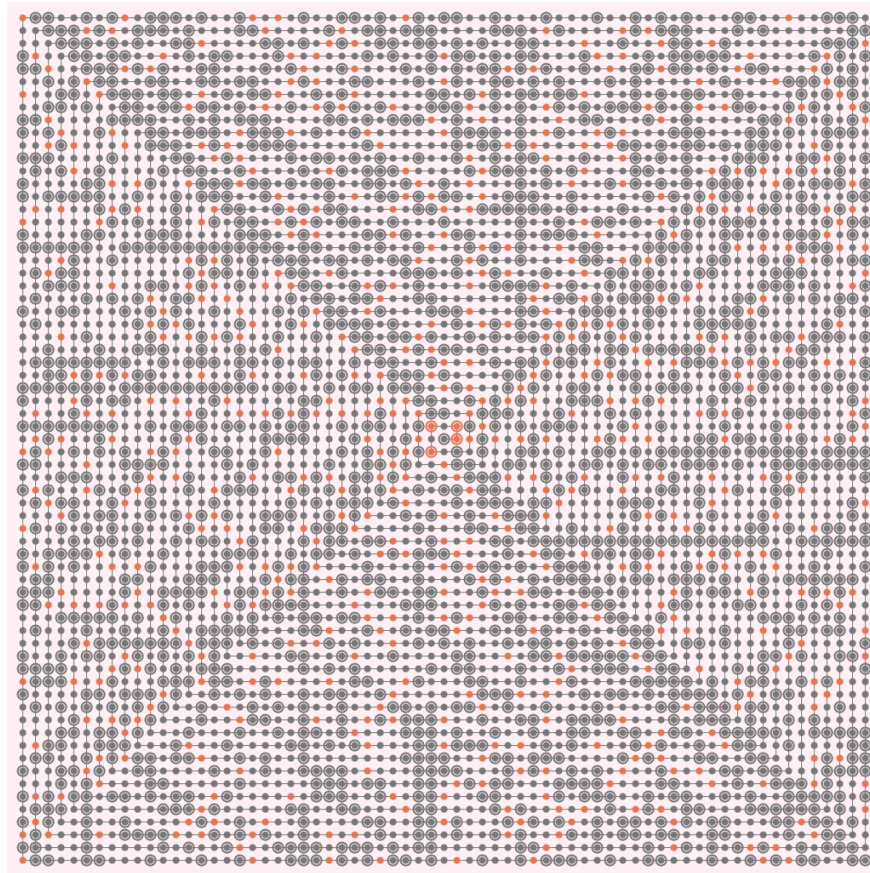
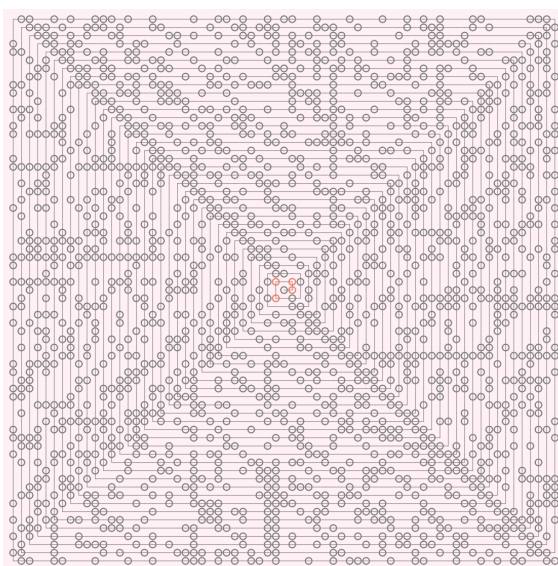
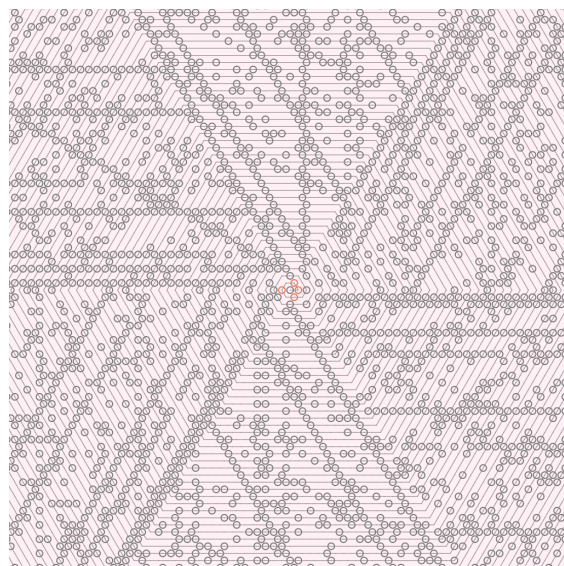


Figure 3.15: Number Spiral



(a) Rectangle Spiral



(b) Hexagonal Spiral

Figure 3.16: Primordial Spiral

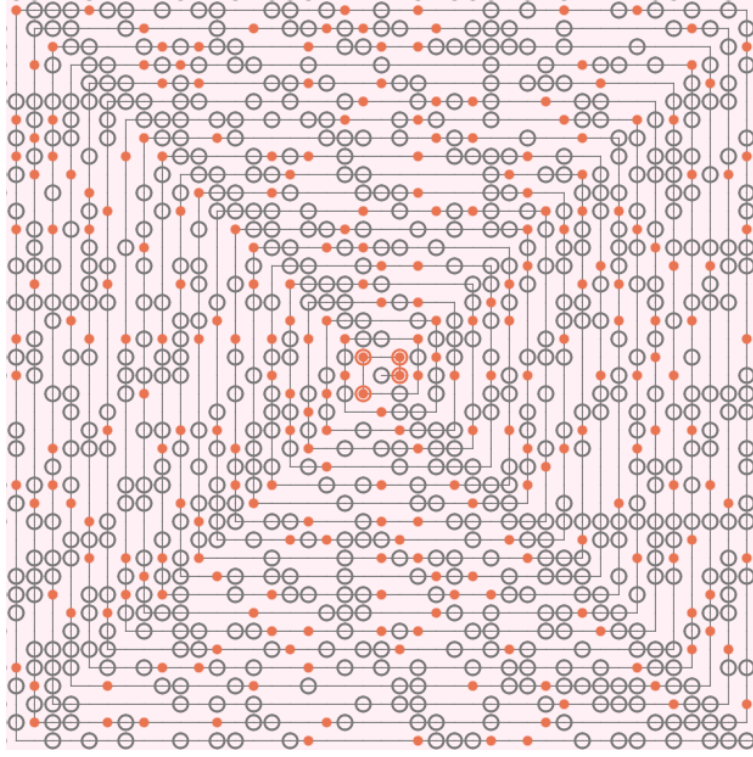


Figure 3.17: Prime and Centrum Numbers of the Number Spiral

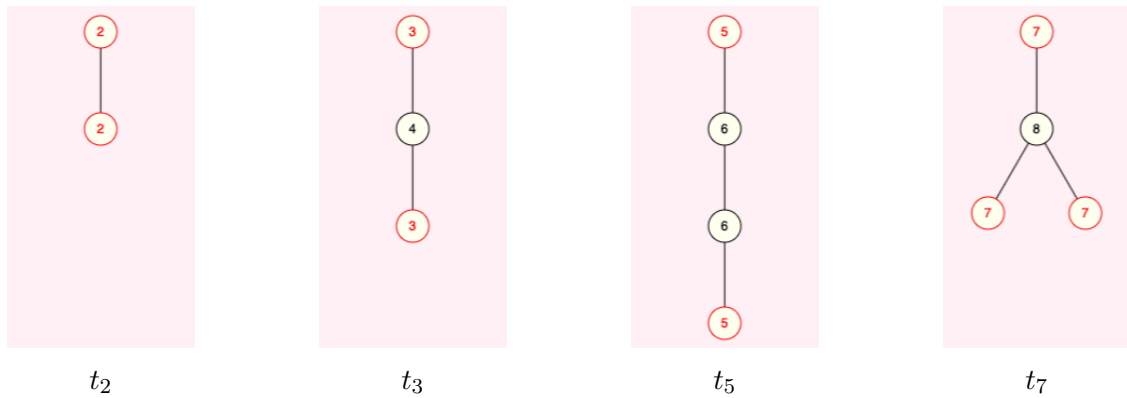
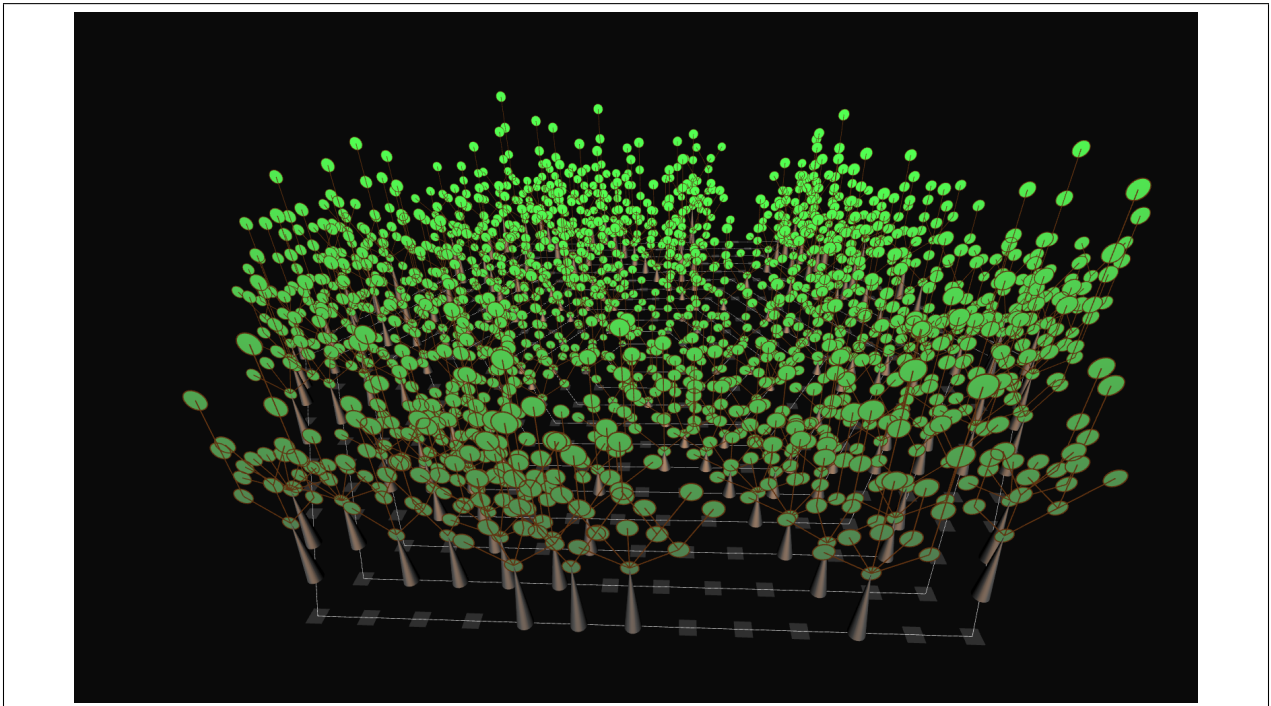


Figure 3.18: Primordial Trees: t_2, t_3, t_5, t_7

pair of cells in the $i \cdot p_j$ matrix symmetric to the main diagonal represent neighboring nodes in a primordial tree. Therefore, the adjacent nodes of the nodes with prime label values (i.e., the cells in the first column) have label values listed in the first row of the matrix. Given that the value of the k_{th} prime function, $p(k)$, increases much faster than the linear value of k , the prime values greater than 7 (i.e., $k > 4$) in the first column will always be larger than their symmetric values in the first row. Thus, the prime value greater than 7 will never be the label value of the centrum node for any primordial tree. \square

By combining the primordial tree structure with the spiral geometric form factor, we have created an online visualization program called “Primordial Garden” (<https://s2.smu.edu/~zizhenc/Project/PrimordialGarden>). This program displays natural numbers (as flat squares) on the spiral in a 3D space, where each centrum number location grows (originates) a primordial tree. The overall view resembles a garden that depicts the entire number theory world as Video 3.4 shows.



Video 3.4: [Primordial Garden](https://s2.smu.edu/~zizhenc/Project/PrimordialGarden)

3.6. Conclusion

We have introduced a graphical partitioning of the entire natural number network into disjoint clusters of number sets each represented by a non-isomorphic directed tree termed “primordial tree”. Each tree is shown to encapsulate the natural numbers of the isomorphic rooted trees of Matula number without the specification of the root. A relation “ $i \cdot p_j \mathbb{R} j \cdot p_i$ ” ($i, j \in \mathbb{N}$), a data structure “ $i \cdot p_j$ matrix” and a graphical depiction “primordial spiral” are discovered based on the primordial tree concept. The potential for application of these discoveries now unseen are hopefully suggested by future works.

A graphical numeral font system is created as a byproduct of the graphical representation of natural numbers, which is also the foundation of the primordial tree.

3.6.1. Contributions of Visualized Algorithm Engineering (VAE)

This entire work is originated and explored from the visualization of natural numbers. This chapter describes the entire journey of our discoveries guided by several VAE works. Initially, we wanted to draw the rooted trees of Matula number as graphical fonts of natural numbers. This guided us to discover the non-isomorphic tree structure, namely the primordial tree. Subsequently, we discovered the integer connectivity relation, the matrix data structure, and the new number spiral.

3.6.2. Future Works

Suggested future works include two aspects: exploration of properties from the primordial tree, $i \cdot p_j$ matrix data structure, and the primordial spiral; establishment and proof of the equivalence relation *integer connectivity* (mentioned in the end of Section [3.5.3.2](#)).

Chapter 4

VISUALIZATION’S ROLE IN ALGORITHM ENGINEERING

4.1. Introduction to Visualized Algorithm Engineering (VAE)

The process of Algorithm Engineering is a multifaceted and intricate undertaking that involves various challenges and potential obstacles, such as implementing, debugging, testing, engineering, and experimentally analyzing algorithmic codes. Thanks to the current computer graphics technology’s ability to convey a large amount of information concisely and humans’ aptitude for processing visual information, visualization plays a crucial role in algorithm engineering. As a result, we propose the term “Visualized Algorithm Engineering (VAE)” to underscore the significance of visualization in this domain. Related concepts in this field include Data/Information Visualization, Algorithm Visualization, Algorithm Animation, and Creative Coding. [12, 34, 63, 64].

VAE refers to the process of designing and creating algorithms using visualizing techniques. It involves the use of graphical tools or creative coding to model and simulate the behavior of algorithms, making it easier for developers to design, test, and optimize their algorithms down to the hardware level. By visualizing algorithms, one can easily identify their strengths and weaknesses and find ways to improve them. It also enables developers or researchers to communicate their ideas more effectively to others, which is exactly what this thesis is about.

4.2. Evaluate VAE on the Two Graph Partitioning Problems

This thesis showcases the applicability of Visualized Algorithm Engineering (VAE) in solving two graph partitioning problems: Backbone Determination in Wireless Sensor Networks (WSNs) and Graphical Partitioning of the Natural Number Network. The former

problem comes from a practical challenge in WSN area (i.e., backbone determination), which is then modeled into a mathematical problem — the partitioning of an Random Geometric Graph (RGG) into multiple disjoint bipartite subgraphs for the goal of each subgraph dominating $(1 - \epsilon)n$ nodes. While the latter is originated from a data visualization for natural numbers (into rooted tree form factor) to a discovery of a graphical partitioning of the entire natural number network.

Visualization plays a critical role in solving both of the problems presented in this thesis. In Chapter 2, we were initially attempting to conduct VAE for the work of Diah Mahjoub’s [46], who proposed using the Smallest-last coloring algorithm to determine backbones in WSNs. After using several sample RGGs to run the created animated graphical program, we discovered that the initial multiple independent color sets resulting from the algorithm had fairly stable sizes with close-packed nodes. This inspired us to recolor the remaining nodes to pair with the initial color sets. The ability to directly observe the performance of running the algorithms also motivated us to optimize and improve the algorithms to achieve linear time efficiency.

In Chapter 3, our initial goal was to visualize the Matula number for teaching purposes. However, we soon noticed the automorphism feature of the Matula number and created a new non-isomorphic tree, known as a “primordial tree”, to cluster all the isomorphic rooted trees of Matula number, each of which maps to a natural number. This allowed us to partition the entire natural number network, leading to a better understanding of the underlying structure.

4.3. Creative Coding

There are numerous techniques and software tools available with various visual capabilities. It is important to note that the Visualized Algorithm Engineering (VAE) does not impose any restrictions on the use of these tools or techniques. Creative coding is a type of programming that is widely used in the field of visual arts [34]. The VAE works presented in this thesis use creative coding techniques, with the objective of creating expressive algo-

rithms and data that go beyond being functional or informative. For example, to evaluate the domination of each resulting subgraphs for the first problem, visualizing a disk to indicate the area each node covers (as Figure 4.1 illustrates) would be more expressive compared to a pure data or any plot chart.

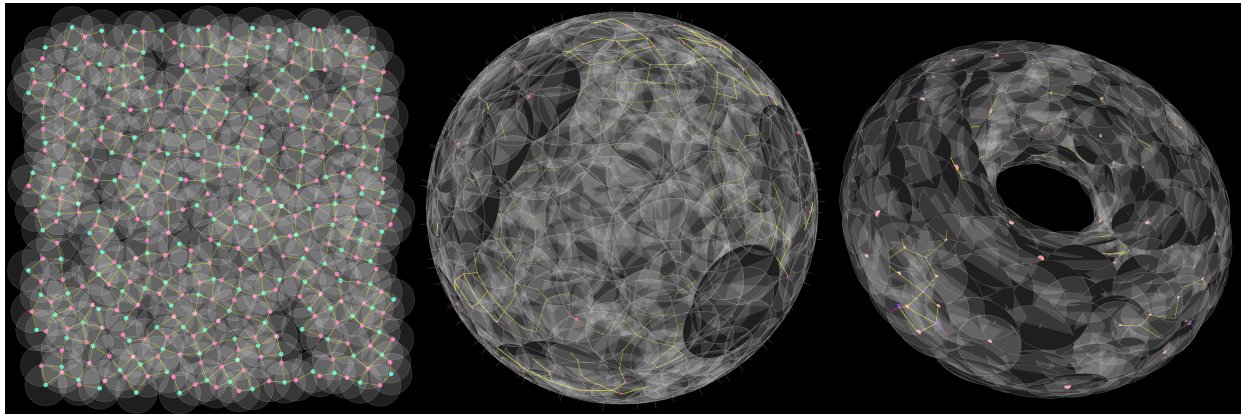


Figure 4.1: Domination of Sample Backbones

Using any software tools to visually “plot” data, we will be limited by the rules or features provided by the tool. In contrast, creative coding allows for embedded visualization with programming. For example, the Processing Foundation offers several languages that are inherently capable of visualizing data while coding, such as the Processing language used in the VAE presented here, which is essentially a visualized version of Java.

4.4. Conclusion

In the past, the diffusion of the use of Algorithm Animation and Algorithm Visualization in Algorithm Engineering encountered numerous obstacles from owing to the absence of visualization/animation systems that facilitated fast prototyping mechanisms [20]. However, with the growing popularity of visualizing techniques such as creative coding, even novice graphic programmers can now employ Algorithm Visualization/Animation for purposes beyond just teaching. This thesis showcases the application of Visualized Algorithm Engineering (VAE) on two graph partitioning problems, using it as a research method to unearth new findings.

BIBLIOGRAPHY

- [1] ABBENA, E., SALAMON, S., AND GRAY, A. *Modern differential geometry of curves and surfaces with Mathematica*. CRC press, 2017.
- [2] AL-KARAKI, J. N., AND KAMAL, A. E. Efficient virtual-backbone routing in mobile ad hoc networks. *Computer Networks* 52, 2 (2008), 327–350.
- [3] AXLER, C. New bounds for the prime counting function $\pi(x)$. *arXiv preprint arXiv:1409.1780* (2014).
- [4] BACH, E., AND SHALLIT, J. *Algorithmic Number theory, volume I*, 1996.
- [5] BADER, G. D., AND HOGUE, C. W. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics* 4, 1 (2003), 1–27.
- [6] BARBEAU, M., BOSE, P., CARMI, P., COUTURE, M., AND KRANAKIS, E. Location-oblivious distributed unit disk graph coloring. *Algorithmica* 60, 2 (2011), 236–249.
- [7] BETTSTETTER, C. On the minimum node degree and connectivity of a wireless multihop network. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing* (2002), pp. 80–91.
- [8] BIGGS, N., LLOYD, E. K., AND WILSON, R. J. *Graph Theory, 1736-1936*. Oxford University Press, 1986.
- [9] BLOUGH, D. M., LEONCINI, M., RESTA, G., AND SANTI, P. The k-neigh protocol for symmetric topology control in ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing* (2003), pp. 141–152.
- [10] BRETÈCHE, R. D. L., AND TENENBAUM, G. Sur certaines équations fonctionnelles arithmétiques. In *Annales de l'institut Fourier* (2000), vol. 50, pp. 1445–1505.
- [11] BRIGHT, L. F., KLEISER, S. B., AND GRAU, S. L. Too much Facebook? An exploratory examination of social media fatigue. *Computers in Human Behavior* 44 (2015), 148–155.
- [12] BROWN, M. H., AND SEDGEWICK, R. Techniques for algorithm animation. *Ieee Software* 2, 1 (1985), 28.
- [13] CARLOS-MANCILLA, M., LÓPEZ-MELLADO, E., AND SILLER, M. Wireless sensor networks formation: approaches and techniques. *Journal of Sensors* 2016 (2016).
- [14] CAROTHERS, N. L. *Real analysis*. Cambridge University Press, 2000.

- [15] CARTIGNY, J., SIMPLOT, D., AND STOJMENOVIC, I. Localized minimum-energy broadcasting in ad-hoc networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)* (2003), vol. 3, IEEE, pp. 2210–2217.
- [16] CHEN, X., AND SHEN, J. Reducing connected dominating set size with multipoint relays in ad hoc wireless networks. In *7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004. Proceedings.* (2004), IEEE, pp. 539–543.
- [17] CHEN, Z., AND MATULA, D. W. Bipartite Grid Partitioning of a Random Geometric Graph. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)* (2017), IEEE, pp. 163–169.
- [18] DAS, B., AND BHARGHAVAN, V. Routing in ad-hoc networks using minimum connected dominating sets. In *Proceedings of ICC'97-International Conference on Communications* (1997), vol. 1, IEEE, pp. 376–380.
- [19] DEL RE, E., PUCCI, R., AND RONGA, L. S. IEEE802. 15.4 wireless sensor network in Mars exploration scenario. In *2009 International Workshop on Satellite and Space Communications* (2009), IEEE, pp. 284–288.
- [20] DEMETRESCU, C., FINOCCHI, I., ITALIANO, G. F., AND NÄHER, S. Visualization in algorithm engineering: Tools and techniques. *Experimental algorithmics: from algorithm design to robust and efficient software* (2002), 24–50.
- [21] DEO, N. *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 2017.
- [22] DUSART, P. Estimates of some functions over primes without R.H. *arXiv preprint arXiv:1002.0442* (2010).
- [23] ELK, S. A problem with the application of Matula’s method of prime numbers and rooted trees for canonical nomenclatures of alkanes. *Graph theory notes (New York)* 18 (1989), 40–43.
- [24] ERDŐS, P., AND HAJNAL, A. On chromatic number of graphs and set-systems. *Acta Math. Acad. Sci. Hungar* 17, 61-99 (1966), 1.
- [25] EULER, L. Variæ observationes circa series infinitas. *Commentarii academiae scientiarum imperialis Petropolitanae* 9, 1737 (1737), 160–188.
- [26] EULER, L. Elementa doctrinae solidorum. *Novi commentarii academiae scientiarum Petropolitanae* (1758), 109–140.
- [27] EVEN, S. *Graph algorithms*. Cambridge University Press, 2011.
- [28] FERDINANDY, B. What’s the difference between a graph and a network.
- [29] GARDNER, M. The remarkable lore of the prime numbers. *Scientific American* 210, 3 (1964), 120.
- [30] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability*, vol. 174. freeman San Francisco, 1979.

- [31] GARG, V., AND JHAMB, M. A review of wireless sensor network on localization techniques. *Int. J. Eng. Trends Technol* 4, 4 (2013), 1049–1053.
- [32] GILBERT, E. N. Random plane networks. *Journal of the society for industrial and applied mathematics* 9, 4 (1961), 533–543.
- [33] GRADY, D. The vision thing: Mainly in the brain. *Discover* 14, 6 (1993), 56–66.
- [34] GREENBERG, I., XU, D., AND KUMAR, D. *Processing: Creative Coding and Generative Art in Processing 2*. Apress, 2013.
- [35] GUPTA, H. P., VENKATESH, T., RAO, S. V., DUTTA, T., AND IYER, R. R. Analysis of coverage under border effects in three-dimensional mobile sensor networks. *IEEE Transactions on Mobile Computing* 16, 9 (2016), 2436–2449.
- [36] GUTMAN, I., IVIC, A., AND ELK, S. Matula numbers for coding chemical structures and some of their properties. *JOURNAL-SERBIAN CHEMICAL SOCIETY* 58 (1993), 193–193.
- [37] HEYMANN, Y. An algorithm for the prime-counting function of primes larger than three. *arXiv e-prints* (2020), arXiv–2002.
- [38] HOCHE, R. G., ET AL. *Nicomachi Geraseni Pythagorei introductionis arithmeticae libri II*. In aedibus BG Tevbneri, 1866.
- [39] KENNICHE, H., AND RAVELOMANANANA, V. Random geometric graphs as model of wireless sensor networks. In *2010 The 2nd international conference on computer and automation engineering (ICCAE)* (2010), vol. 4, IEEE, pp. 103–107.
- [40] KNUTH, D. *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2. Addison-Wesley, Reading, Mass, 1981.
- [41] KOPYTOV, N., AND MITYUSHOV, E. The method for uniform distribution of points on surfaces in multi-dimensional Euclidean space. *Intellectual Archive* (2012).
- [42] KOTNIK, T. The prime-counting function and its analytic approximations: $\pi(x)$ and its approximations. *Advances in Computational Mathematics* 29, 1 (2008), 55–70.
- [43] KUMAR, S., LAI, T. H., AND BALOGH, J. On k -coverage in a mostly sleeping sensor network. In *Proceedings of the 10th annual international conference on Mobile computing and networking* (2004), pp. 144–158.
- [44] LICK, D. R., AND WHITE, A. T. k -Degenerate graphs. *Canadian Journal of Mathematics* 22, 5 (1970), 1082–1096.
- [45] MAC DONALD, V. H. Advanced mobile phone service: The cellular concept. *The bell system technical Journal* 58, 1 (1979), 15–41.
- [46] MAHJOUB, D. *Efficient redundant backbones for coverage and routing in wireless sensor networks*. PhD thesis, Southern Methodist University, 2011.
- [47] MAHJOUB, D., AND MATULA, D. W. Experimental Study of Independent and Dominating Sets in Wireless Sensor Networks Using Graph Coloring Algorithms. In *International Conference on Wireless Algorithms, Systems, and Applications* (2009), Springer, pp. 32–42.

- [48] MAHJOUB, D., AND MATULA, D. W. Building $(1 - \varepsilon)$ dominating sets partition as backbones in wireless sensor networks using distributed graph coloring. In *International Conference on Distributed Computing in Sensor Systems* (2010), Springer, pp. 144–157.
- [49] MARSAGLIA, G., ET AL. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics* 43, 2 (1972), 645–646.
- [50] MATULA, D. W. A natural rooted tree enumeration by prime factorization. In *SIAM REVIEW* (1968), vol. 10, SIAM PUBLICATIONS 3600 UNIV CITY SCIENCE CENTER, PHILADELPHIA, PA 19104-2688, p. 273.
- [51] MATULA, D. W., AND BECK, L. L. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)* 30, 3 (1983), 417–427.
- [52] MATULA, D. W., AND CHEN, Z. Precise and Concise Graphical Representation of the Natural Numbers. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)* (2019), IEEE, pp. 100–103.
- [53] MATULA, D. W., MARBLE, G., AND ISAACSON, J. D. Graph coloring algorithms. In *Graph theory and computing*. Elsevier, 1972, pp. 109–122.
- [54] MATULA, D. W., AND SOKAL, R. R. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis* 12, 3 (1980), 205–222.
- [55] NANDI, A., AND KUNDU, S. Optimal transmit power and energy level performance of random WSN in Rayleigh fading channel. In *2011 2nd International Conference on Computer and Communication Technology (ICCCCT-2011)* (2011), IEEE, pp. 556–561.
- [56] OLASCUAGA-CABRERA, J. G., LÓPEZ-MELLADO, E., MENDEZ-VAZQUEZ, A., AND RAMOS-CORCHADO, F. F. A self-organization algorithm for robust networking of wireless devices. *IEEE Sensors Journal* 11, 3 (2010), 771–780.
- [57] PENROSE, M., ET AL. *Random geometric graphs*, vol. 5. Oxford university press, 2003.
- [58] POE, W. Y., AND SCHMITT, J. B. Node deployment in large wireless sensor networks: coverage, energy consumption, and worst-case delay. In *Asian internet engineering conference* (2009), pp. 77–84.
- [59] PRASAD, K. D., AND MURTY, S. Wireless Sensor Networks—A potential tool to probe for water on Moon. *Advances in Space Research* 48, 3 (2011), 601–612.
- [60] RANGANATHAN, P., AND NYGARD, K. Time synchronization in wireless sensor networks: a survey. *International journal of ubicomp* 1, 2 (2010), 92–102.
- [61] RAVELOMANANA, V. Extremal properties of three-dimensional sensor networks with applications. *IEEE Transactions on Mobile Computing* 3, 3 (2004), 246–257.
- [62] REAS, C., AND FRY, B. *Processing: a programming handbook for visual designers and artists*. Mit Press, 2007.
- [63] SHAFFER, C. A., COOPER, M. L., ALON, A. J. D., AKBAR, M., STEWART, M., PONCE, S., AND EDWARDS, S. H. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)* 10, 3 (2010), 1–22.

- [64] SHEWAN, D. Data is Beautiful: 7 Data Visualization Tools for Digital Marketers. *Business2Community. com*. Archived from the original on 12 (2016).
- [65] SIMSON, R., ET AL. *The elements of Euclid*. Desilver, Thomas, 1838.
- [66] SNIJDERS, C., MATZAT, U., AND REIPS, U.-D. “Big Data”: big gaps of knowledge in the field of internet science. *International journal of internet science* 7, 1 (2012), 1–5.
- [67] SOHRABY, K., MINOLI, D., AND ZNATI, T. *Wireless sensor networks: technology, protocols, and applications*. John wiley & sons, 2007.
- [68] TSENG, Y.-C., NI, S.-Y., CHEN, Y.-S., AND SHEU, J.-P. The broadcast storm problem in a mobile ad hoc network. *Wireless networks* 8, 2 (2002), 153–167.
- [69] WAGNER, D., AND WATTENHOFER, R. *Algorithms for sensor and ad hoc networks: advanced lectures*, vol. 4621. Springer, 2007.
- [70] WAN, P.-J., ALZOUBI, K. M., AND FRIEDER, O. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications* 9, 2 (2004), 141–149.
- [71] WANG, Y., WANG, W., AND LI, X.-Y. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing* (2005), pp. 2–13.
- [72] WEI, C., YANG, J., GAO, Y., AND ZHANG, Z. Cluster-based routing protocols in wireless sensor networks: A survey. In *Proceedings of 2011 International Conference on Computer Science and Network Technology* (2011), vol. 3, IEEE, pp. 1659–1663.
- [73] WERNER-ALLEN, G., LORINCZ, K., RUIZ, M., MARCILLO, O., JOHNSON, J., LEES, J., AND WELSH, M. Deploying a wireless sensor network on an active volcano. *IEEE internet computing* 10, 2 (2006), 18–25.
- [74] WU, J., AND LI, H. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications* (1999), pp. 7–14.
- [75] YE, J. *Computing exact bottleneck distance on random point sets*. PhD thesis, Virginia Tech, 2020.
- [76] ZHANG, H., HOU, J. C., ET AL. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc Sens. Wirel. Networks* 1, 1-2 (2005), 89–124.
- [77] ZHAO, J., YAĞAN, O., AND GLIGOR, V. Connectivity in secure wireless sensor networks under transmission constraints. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2014), IEEE, pp. 1294–1301.