# Dispersion of Mobile Robots on Triangular Grid

by

**Yash Kirankumar Joshi**
**202111027**

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY


May, 2023

## Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Yash Kirankumar Joshi

## Certificate

This is to certify that the thesis work entitled Dispersion of Mobile Robots on Triangular Grid has been carried out by Yash Kirankumar Joshi for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.


Prof. Supantha Pandit
Thesis Supervisor

# Acknowledgments

I want to take this opportunity to express my sincere appreciation to all those who have contributed to the completion of this thesis. Their unwavering support, guidance, and encouragement have played a vital role in shaping my research journey, and I am profoundly grateful to them.

First and foremost, I extend my deepest gratitude to my supervisor, Prof. Supantha Pandit, whose expertise, patience, and constant motivation have been invaluable throughout this entire process. Your insightful feedback and constructive criticism have pushed me to explore new avenues in my research, ultimately enhancing the quality of this thesis.

I also want to express my heartfelt appreciation to my friends and colleagues for their unwavering support and encouragement. Their friendship, engaging discussions, and shared experiences have made this journey more enjoyable and memorable. Their belief in my abilities has motivated me during challenging times, and I am fortunate to have such remarkable individuals in my life.

Lastly, my deepest gratitude goes to my family for their unwavering support, love, and belief in my abilities. Their sacrifices, understanding, and encouragement have given me the strength and inspiration to pursue my educational goals. Their presence has been instrumental in my achievements, and I am truly blessed to have such a loving and supportive family.

In conclusion, I am profoundly grateful to all those who have played a part in shaping my academic journey and making this thesis a reality. Thank you for your unwavering support, guidance, and encouragement. Your impact on my personal and professional growth is immeasurable, and I am truly fortunate to have had you alongside me.

# Contents

# Abstract

The utilization of mobile robots to solve global problems in a distributed manner is an intriguing and novel approach to problem solving. In this paradigm, each robot operates independently while collaborating with other robots to achieve a goal that would be impossible to achieve using a centralised global approach. This method can be used to simulate a variety of real-world problems, such as toxic hazard cleanup, large-scale maze exploration, and collective gathering in a single location.

The Dispersion problem on a Triangular Grid with a particular set of configurations is introduced in this thesis. We present a five-Round algorithm that can solve the problem in $O(max(H_G, W_G))$ time rounds, using only $O(\log k)$ bits for the mobile robots, where $H_G$ is height of grid, $W_G$ is width of grid and k is number of the mobile robots. Our solution provides an optimal solution in terms of both time and memory complexity for Dispersion on a given configuration of a triangular grid.

# List of Figures

# CHAPTER 1
# Introduction

## 1.1 Distributed Computing

We have seen extraordinary growth in distributed systems and networks over the last two decades, and distributed computing now incorporates many activities in today's computer and telecommunications environment. Almost every sizeable computerized system in use today is distributed to some extent. Distributed computing applications range from computer and communications networks to distributed database systems, significant financial and economic systems, and industrial control mechanisms. One of the most challenging issues encountered by the application's designers and operators in all of those applications is maintaining the seamless, coordinated, and non-interfering working of multiple remote processors.

One of the significant areas of recent study in distributed network algorithms is a better understanding and management of the locality problem. Most classic network algorithms do not consider locality or leverage it nontrivially. As networks become more prevalent, using several traditional protocols for conducting various network control and management activities becomes significantly more difficult. On the other hand, such global knowledge is only seldom required, and many supposedly global control tasks can be accomplished efficiently while allowing processors to know more about their immediate surroundings and less about the rest of the world. Furthermore, when contemplating a local subtask that only involves vertices in a limited network section, one would prefer it to be executed at a cost proportional to its locality level.

Distributed computing refers to situations in which several processors placed at separate locations must operate non-interfering and cooperatively. Each processor has some autonomy: it runs its own protocol on private hardware and frequently has independent tasks. Nonetheless, the processors must share specific common resources and information, and some coordination is required to ensure

the proper fulfillment of their unique duties.

The utilization of mobile robots to solve global problems in a distributed manner is a unique and intriguing approach to problem-solving. In this paradigm, each robot operates independently, yet collaborates with other robots to accomplish a goal that would be unattainable through a centralized global approach. This approach can be applied to model numerous real-world problems, including toxic hazard cleanup, large-scale maze exploration, and collective gathering at a single location.

This approach has practical applications in the context of self-driving electric cars and charging stations. Recharging an electric vehicle takes time, and when multiple charging stations are nearby, it is more efficient to locate the nearest available station rather than wait for a specific one. As self-driving cars continue to become more commonplace, it is advantageous to devise strategies to automate this process and leverage knowledge about a broader range of public charging stations.

## 1.2 What is Dispersion?

The problem at hand involves the placement of $n$ robots within a graph comprising $n$ nodes. Initially, the robots are positioned arbitrarily across the nodes, and their task is to coordinate their movements effectively. The objective is to achieve a final configuration where each node of the graph is occupied by exactly one robot, requiring them to work together to accomplish this goal.

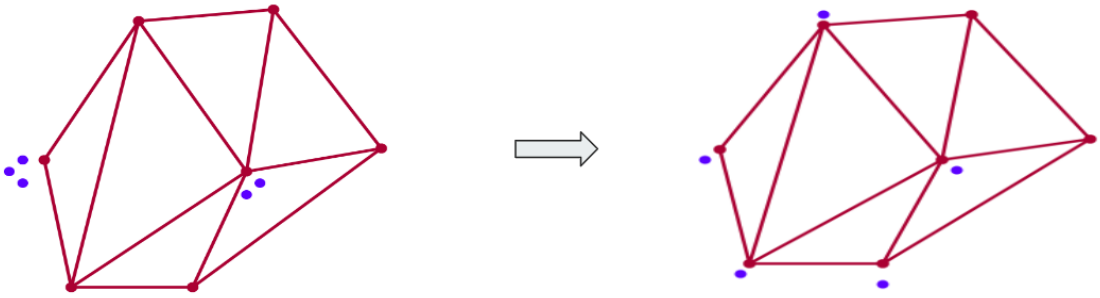

Figure 1.1: Dispersion on the graph

We identify significant linkages between dispersion and other related issues such as scattering, mobile robot exploration, and load balancing. The problem of $n$-robot collaborative graph exploration, in which the goal is for the group of n robots to explore the whole graph in the fewest rounds feasible, can be compared

to that of dispersion. It is important to note that, given the same circumstances, any dispersion problem solution can be converted into a solution for $n$-robot exploration. As a result, $n$-robot exploration can benefit directly from the findings and understandings obtained through studying dispersion. Results of exploration often fall into one of two categories: reducing the number of rounds necessary to complete exploration or proving that exploration is feasible even with limited memory or communication resources.

The problem of robot scattering or uniform-deployment on graphs, where the goal is to evenly disperse robots among the nodes of the graph, demonstrates a strong correlation with dispersion. In particular, assuming a similar modelling approach, the dispersion problem becomes equivalent to the uniform-deployment problem when the number of robots is equal to the number of nodes in the network. Although they have different initial setups and restrictions, both issues ultimately aim to achieve a balanced distribution of robots around the graph.

Similar to load balancing on graphs, where nodes initially carry various loads and attempt to transfer the weight across edges until an approximately equal distribution is attained, the problem of dispersion involves nodes carrying varying loads. When it comes to load balancing, the issue is equivalent to dispersion if the burdens have memory and computing power and are labelled, but the nodes are unlabeled and lack these resources. Both issues entail redistributing items (burdens or robots) among the graph's nodes while taking restrictions and resource availability into account. Dispersion concentrates on achieving a distinct presence of robots at each node in the graph, whereas load balancing primarily concentrates on achieving an equitable distribution of loads.

## 1.3 Thesis Objective

The Dispersion problem, which is centred on a triangular graph $G(L)$ with $N$ nodes and $L$ layers, is what we aim to solve. At first, the nodes of $G$ are occupied by $k$ mobile robots that are randomly positioned. The main goal of these robots is to autonomously move to a configuration where each robot resides in a different node of $G$. Our goal in solving this issue is to optimise two vital performance indicators: time, which is gauged by the quantity of rounds needed, and memory, which is determined by the amount of bits held at each robot. The final objective is to create methods that reduce the robots' memory usage and the amount of time needed for dispersion.

## 1.4 Thesis Outline

The proposed thesis structure encompasses several key chapters that adhere to a common and well-established format. It consists of the following components:

### 1.4.1 Chapter 2: Literature Survey

This chapter serves as a thorough analysis and synthesis of pertinent academic research findings and literature with regard to the subject matter of this thesis. Its main goal is to create a full understanding of the state of the field's knowledge at the moment. This chapter aims to provide a thorough review of the topic and add to the body of knowledge in the field by looking at existing theories, techniques, and empirical studies.

### 1.4.2 Chapter 3: Model and Definitions

The conceptual framework and definitions that serve as the thesis's cornerstone are the main topics of Chapter 3. It seeks to make the theoretical frameworks, ideas, and variables essential to the study more understandable. This chapter helps readers comprehend the theoretical foundations upon which the analysis and conclusions of the thesis are built by clearly defining these important components. It creates a strong foundation for the later analysis and interpretation of the research results.

### 1.4.3 Chapter 4: Algorithm

The design of an algorithm employed in the study are the focus of Chapter 4. It gives a thorough account of the steps taken in the methods, procedures, or strategies used to answer the research questions or accomplish the desired goals. In order for readers to duplicate or comprehend the research process, the chapter strives to provide a thorough and understandable description. Visual aids like pseudocode, flowcharts, or diagrams may be used to help the reader understand. The chapter acts as a manual, giving readers all the details they need to precisely follow and duplicate the study technique.

### 1.4.4 Chapter 5: Conclusions

We deliver a thorough summary of the research conducted and the findings from the study in the last chapter, Chapter 5. The main conclusions, their ramifica-

tions, and how they relate to the study's goals or questions are all summarised in this part. The author also outlines potential directions for future research and acknowledges any limitations observed throughout the study. This chapter makes sure that readers have a thorough comprehension of the research contributions and their significance by briefly summarising the entire thesis. It acts as a summative reflection on the research process, stressing the most important lessons learned and putting an end to the thesis.

# CHAPTER 2
# Literature Survey

According to the [15, 16], the challenge of distributing autonomous mobile robots to achieve an even distribution across a particular region has garnered a lot of interest. Recently, Augustine and Moses Jr. [1] used the structure of graphs to approach this issue. The following is a description of their formulation: The robots autonomously alter their placements to reach a final configuration in which each robot occupies a different node within the graph, starting with an arbitrary beginning configuration in which $k \leq n$ robots are positioned on the nodes of an $n$-node graph. Beyond its own merits, this problem is significant because it is strongly related to a number of other thoroughly researched difficulties in autonomous robot coordination, such as exploration, scattering, load balancing, covering, and self-deployment [1, 18]. The examination of the dispersion problem is vital in furthering our understanding of more general difficulties in the area of autonomous robot coordination since these related problems have similar themes and goals. Understanding how to use the resource-constrained robots to complete some huge tasks in a distributed manner is one of the most important elements of mobile-robot research [10, 11].

The dispersion problem is closely related to the issue of mobile robot graph exploration. Numerous studies [2, 4, 8, 13, 19] show that graph exploration has gotten a lot of attention in the literature and may be applied to both specified and arbitrary graphs. With a memory need of $\Theta(D \log \Delta)$ bits, a robot was shown to be able to navigate an anonymous graph using an exploration technique. The runtime complexity of the algorithm is $O(\Delta^{D+1})$ [13]. Cohen et al. [4] suggested two techniques in circumstances where the graph nodes have memory. In comparison to the second technique, which uses $O(\log \Delta)$ bits at the robot and 1 bit at each node, the first algorithm uses $O(1)$ bits at the robot and 2 bits at each node.Both techniques have an $O(m)$ runtime complexity and an $O(mD)$ preprocessing time. The trade-off between exploration time and robot count is examined in [19]. Collective robot exploration in the setting of tree structures is looked at in [12]. The

scattering of $k$ robots within graphs, which has been investigated in prior works, is also a related issue to dispersion. Numerous studies have been done on the scattering problem using rings [9, 21] and grids [3]. A $\Theta(\sqrt{n})$-algorithm approach for achieving uniform scattering in a grid [7] was presented by Poudel and Sharma [20]. The load balancing problem, which requires dividing a given load among a number of processors (nodes), also shares ties with the dispersion problem. According to earlier studies [22, 5], this load balancing issue has drawn a lot of interest in the study of graphs. We advise referring to [10, 11] for more recent discoveries and advancements in these fields. As the major subject of these studies, the gathering problem has been examined in relation to a number of fundamental network topologies. Recent years have seen a great amount of research into the gathering of mobile robots. Mobile robots need to be directed in the direction of the gathering place in order to solve the problem of gathering. By investigating a number of key network topologies, researchers expect to get insight into the complexities and nuances involved with creating successful gathering settings. In [14], with any initial configuration, they have described the gathering issue for robots with 1-hop vision on a triangular grid. They are informed that the gathering process on an unending triangular grid must run for $\Omega(n)$ epochs when there are $n$ robots. As a result, they offered an algorithm that, when used with a semi-synchronous scheduler, gathers all the robots to a single location in $O(n)$ epochs. One epoch is a period of time during which each robot has been turned on at least once.

In [17] by Klasing et al. and [6] by D'Angelo et al. for two studies that looked at the gathering issue in various network configurations. The ring network was the focus of Klasing et al.'s study, which showed that assembling all entities in one place is impossible without being able to tell if many entities are occupying the same position. In other words, it is impossible to achieve a gathering configuration on a ring network without multiplicity detection. D'Angelo et al. defined the gathering problem and looked at several scenarios in their study on tree and finite grid networks. They notably looked at the effects of periodic layouts and configurations with symmetry lines running through the grid's edges. They discovered that gathering is not achievable in certain network topologies even with global-strong multiplicity detection.

# CHAPTER 3
# Model and Definitions

## 3.1 Graph

A graph is a mathematical concept used to depict connections or relationships between different objects. It consists of two main elements: vertices (also called nodes) and edges. Vertices represent the objects themselves, while edges represent the connections or links between the objects.In graphs, vertices and edges can possess additional attributes or properties. For instance, edges can have weights assigned to them, indicating the strength or distance of the relationship between the vertices. These attributes are helpful in solving specific problems or analyzing the characteristics of the graph.

**Definition 1** (Graph)**.** *A graph is a mathematical structure composed of vertices (also called nodes) and edges. It is represented by the ordered pair G = (V, E), where V represents the set of vertices and E represents the set of edges.*

Vertices represent distinct objects or entities, while edges represent the connections or relationships between them. These relationships can be direct or indirect and can represent various types of associations, dependencies, or interactions. Graphs can be visualized as diagrams, with vertices represented as points or circles, and edges represented as lines or arcs that connect the vertices. The arrangement of vertices and edges in a graph provides valuable insights into the structure and relationships of the system or problem being modeled. Graphs can possess additional characteristics. For instance, edges can be assigned weights to indicate the strength, distance, or cost associated with the connections. Graphs can also be directed, where edges have specific orientations, or undirected, where edges represent symmetric relationships.

Graphs serve as a powerful tool for studying and analyzing complex systems across various fields such as computer science, social networks, transportation networks, and biology. They allow researchers and analysts to comprehend the

connectivity, patterns, and behaviors of these systems, enabling the development of efficient algorithms, network analysis methods, and problem-solving strategies.

**Definition 2** (Degree)**.** *The degree of a vertex in a graph refers to the number of edges that are incident to that vertex. The degree of a vertex v is denoted as* $deg(v)$.

The maximum degree of a graph G is represented by $\Delta(G)$, which corresponds to the highest degree among all the vertices in the graph. It provides valuable information about the connectivity and structural characteristics of the graph. Graphs can be categorized into different types based on their characteristics. Some common types include:

1. Directed Graph : In a directed graph, each edge has a specific direction associated with it. The connection between two vertices is one-way, and the edge is represented by an arrow.

2. Undirected Graph: In an undirected graph, edges have no inherent direction. The relationship between vertices is symmetrical, and the edge is typically represented by a line or curve.

3. Weighted Graph: A weighted graph assigns numerical values, known as weights, to edges. These weights can represent various properties, such as cost, distance, or capacity associated with the connection between vertices.

4. Connected Graph: A connected graph is one where there is a path between any two vertices. In simpler terms, it is possible to reach any vertex from any other vertex by following a sequence of edges.

5. Complete Graph: A complete graph is one where there is an edge between every distinct pair of vertices. In a complete graph with $n$ vertices, there are $\frac{n(n-1)}{2}$ edges.

## 3.2 Grid

We will now present a definition for the finite triangular grid, which is analogous to the definition of the infinite triangular grid presented in [14].

**Definition 3** (Finite Triangular Grid)**.** *Let G be a geometric graph represented by G = (V, E), where V is a set of vertices and E is a set of edges. The finite triangular grid*

*G is defined within a bounded region of the Euclidean plane $\mathbb{R}^2$. The vertices of G are positioned according to the following coordinates:*

$$V = \{(k, \frac{\sqrt{3}}{2}i) : k \in \mathbb{Z},\ i \in 2\mathbb{Z},\ 0 \leq k \leq K,\ 0 \leq i \leq I\} \bigcup \{(k + \frac{1}{2}, \frac{\sqrt{3}}{2}i) : k \in \mathbb{Z},\ i \in 2\mathbb{Z} + 1,\ 0 \leq k \leq K,\ 0 \leq i \leq I\},$$

*where K and I are positive integers representing the dimensions of the grid. For two vertices u and v in G, u and v are considered adjacent if and only if the Euclidean distance between them is equal to 1 unit.*

In a finite triangular grid, the coordinates of the vertices are arranged based on a triangular lattice structure. The vertices are positioned in the Euclidean plane $\mathbb{R}^2$ according to a specific pattern. The coordinates of the vertices can be expressed using two integers, $(k, i)$, where $k$ represents the horizontal position and $i$ represents the vertical position of a vertex. These integers correspond to the indices of the vertices in the grid. For the first set of vertices, the coordinates are given by:

$$\{(k, \frac{\sqrt{3}}{2}i) : k \in 2\mathbb{Z},\ i \in \mathbb{Z},\ 0 \leq k \leq K,\ 0 \leq i \leq I\}$$

corresponds to the vertices in rows where the vertical index $i$ is an even number ($2\mathbb{Z}$). These vertices have an even spacing along the vertical axis ($\frac{\sqrt{3}}{2}i$). The second set of vertices is given by:

$$\{(k + \frac{1}{2}, \frac{\sqrt{3}}{2}i) : k \in 2\mathbb{Z} + 1,\ i \in \mathbb{Z},\ 0 \leq k \leq K,\ 0 \leq i \leq I\}$$

In this case, represents the vertices in rows where the vertical index $i$ is an odd number ($2\mathbb{Z} + 1$). These vertices are positioned halfway between the columns of the even rows, resulting in a staggered arrangement. The parameters $K$ and $I$ define the maximum values for the horizontal index $k$ and the vertical index $i$, respectively, ensuring that the grid has finite dimensions. By incorporating these coordinates into the definition, the vertices of the finite triangular grid are correctly positioned, enabling the establishment of adjacency relationships and the construction of the corresponding geometric graph.

**Definition 4** (Layer)**.** *A layer 'l' in a graph is defined as a set of points that share the same y-coordinate value. In other words, given a graph with points represented as $(x_i, y_i)$, a layer consists of all points $(x_i, y)$ in the graph, where y is a constant value that is equal for all points within the layer.*

The number of layers in graph G, can be denoted as |L| = k, where the layers L of graph G is set layers $l_0, l_1, ..., l_k$.

In the thesis, the terms "*nodes*" and "*vertices*" are used interchangeably to refer to the individual elements within the triangular grid, which represent the fundamental units of the graph structure. We have classified these nodes of the triangular grid into three distinct categories. As shown in Fig. 3.1,

**Definition 5** (Internal Node). *A node v has $deg(v) = 6$ is called internal node in the triangular grid, This means that it is connected to exactly six neighboring nodes.*

**Definition 6** (Boundary Node). *A node v has $deg(v) < 6$ is called boundary node in the triangular grid, this is a node that lies on the boundary of the grid.*

**Definition 7** (Corner Node). *A node v is called corner node if:*

1. *Node v has $deg(v) = 2$.*
2. *Node v has $deg(v) = 3$ and it does not have any adjacent node u such that $deg(u) = 5$.*

By categorizing the nodes of the triangular grid based on their degrees, we distinguish between internal nodes with a degree of 6, boundary nodes with fewer than six neighbors, and corner nodes that are endpoints of the boundary. These definitions provide a clear understanding of the different types of nodes within the triangular grid based on their connectivity and placement.

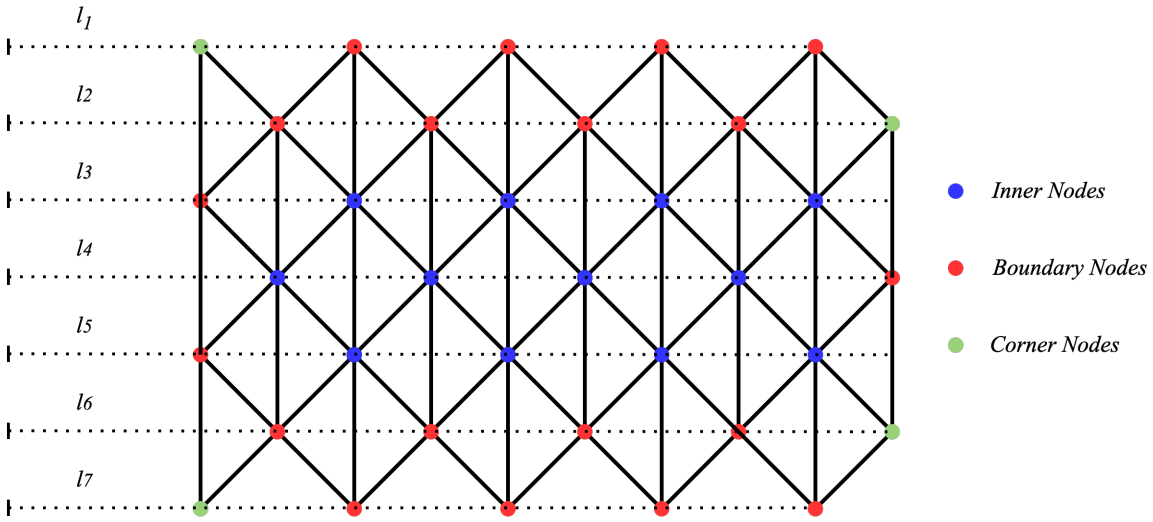


Figure 3.1: Classification of the nodes and layers

Let L is the number of layers of a triangular grid vertically. In this paper, we have taken the configuration of a triangular grid as $G(L)$.

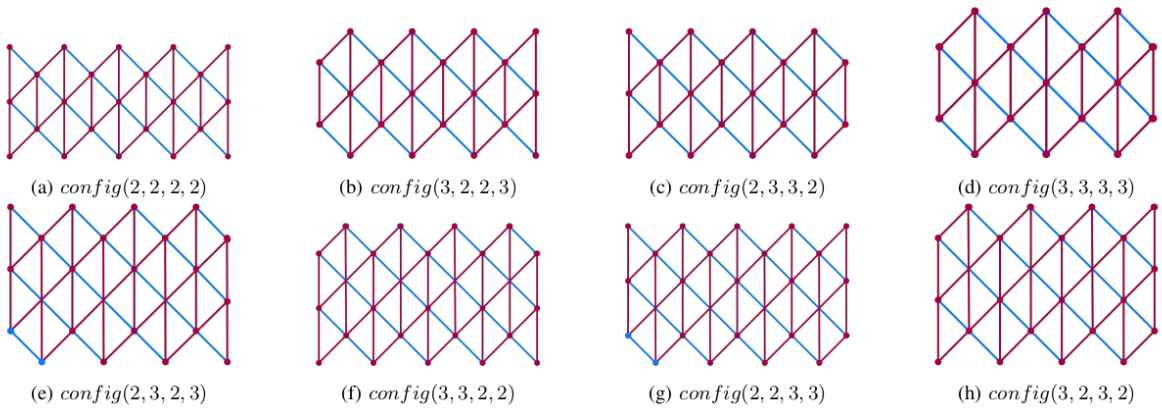


Figure 3.2: All possible configurations of the triangular grid

There are eight possible configurations for a triangular grid which are shown in figure 3.2.

**Definition 8** (Configuration)**.** *For the triangular grid G, a configuration is a specific arrangement or state of the grid, determined by the degrees of its corner nodes. It is denoted as config(a, b, c, d), where 'a,' 'b,' 'c,' and 'd' represent the degrees of the corner nodes of the triangular grid G. Specifically:*

- *'a' represents the degree of the top-left corner node in the configuration.*
- *'b' represents the degree of the top-right corner node in the configuration.*
- *'c' represents the degree of the bottom-right corner node in the configuration.*
- *'d' represents the degree of the bottom-left corner node in the configuration.*

**Lemma 1.** *For any triangular grid G(l), where l represents the number of layers, there exist 8 possible configurations. Among these configurations, only 5 are unique.*

*Proof.* For triangualr grid G(l), let's denote configuration as $config(a, b, c, d)$. As per definition of corner nodes there are only 2 possible degrees for conrers nodes - 2 and 3. Now there are 16 combinations are possible using these two degrees. Now we need to check that how many configurations are possible out of these 16 combinitions.

By the definition of triangular grid, it lies on $\mathbb{R}^2$ which has the dimensions $K$ and $I$. So there is no possibility that corner nodes have the same degree with odd occurrences. Using this fact, there are eight configurations are possible, which are:

1. $config(2, 2, 2, 2)$
2. $config(2, 2, 3, 3)$

3. $config(2,3,2,3)$
4. $config(3,2,3,2)$
5. $config(3,3,2,2)$
6. $config(3,2,2,3)$
7. $config(2,3,3,2)$
8. $config(3,3,3,3)$

Among these eight configurations, three pairs have mirror configurations, meaning they are symmetric or identical when reflected or rotated. These pairs are:

- $cofig(2,3,2,3)$ and $(3,2,3,2)$
- $config(2,2,3,3)$ and $(3,3,2,2)$
- $config(2,3,3,2)$ and $(3,2,2,3)$

Therefore, out of the eight possible configurations, only (8 - 3) = 5 unique configurations exist. □

## 3.3 Robots

The robots in the system possess three distinct characteristics. Firstly, they are autonomous, indicating that there is no centralized control governing their actions. Each robot operates independently, making decisions based on local information and their own internal algorithms without relying on a central authority.

It should also be observed that the dispersion problem robots demonstrate homogeneity, i.e., they all follow the same deterministic method. The coordination and communication processes amongst the robots are made easier by the uniformity of algorithmic behaviour, which guarantees consistency. The robots can efficiently cooperate and work together to accomplish the dispersion target by adhering to a standardised strategy.

Finally, it is crucial to note that each robot in the dispersion problem is given a unique identifier (ID). Individual robots within the system can be identified and tracked using these special IDs. The robots can successfully coordinate, communicate, and manage data because they have allocated IDs. Additionally, the distinctive IDs allow for the monitoring of particular robot behaviours or tasks, which

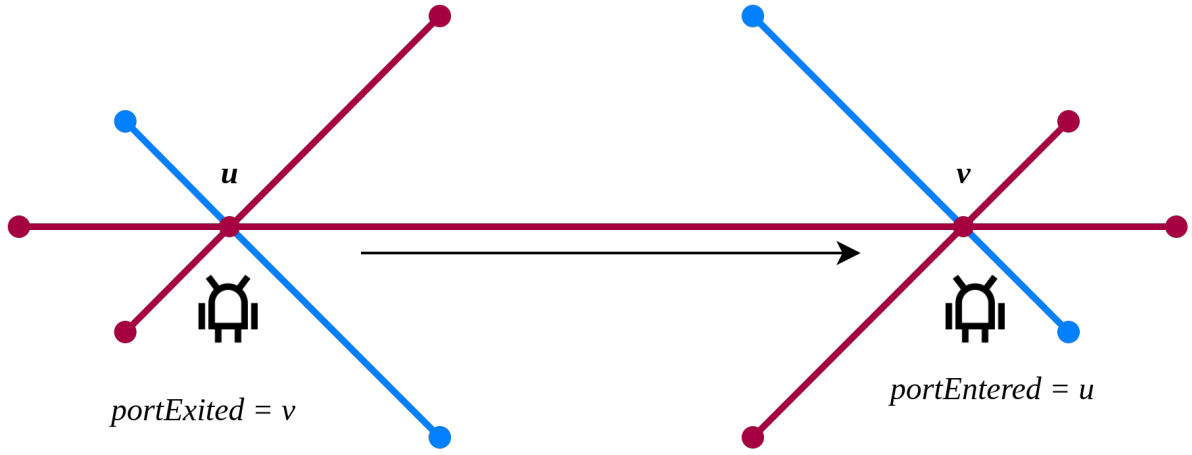


Figure 3.3: Movement of the robot

offers important insights into the overall dynamics and performance of the system.

The robots are positioned on the corners of a triangle grid designated as $G$ in the context of the dispersion problem. It is interesting to observe that the robots do not have a common understanding of a global coordinate system. Instead, each robot has its own distinct localised coordinate system, with the origin or reference point being the robot itself. Each robot also follows its own specific handedness, which adds to the uniqueness of their local coordinate systems. The robots may work autonomously while preserving their own reference frames for navigation and spatial orientation by using this decentralised approach to coordinate systems.

The robots function in the $k$-hop visibility model under limited visibility conditions, with a particular emphasis on the scenario where $k$ is set to 1. Each robot in this example, denoted by the letter $r$ has a limited field of vision. In particular, a robot can only see the six nearby vertices that are directly related to it when it is placed on a vertex $v$ located within the triangular grid $G$. The robot can only see the immediately neighbouring vertices in the grid structure since the sight range is limited to a single hop distance. The robot's vision and awareness of its immediate surroundings are shaped by the confined visibility framework, which limits the robot's knowledge to the surrounding area.

The one-hop visibility model with $(k = 1)$ serves as the basis for the analysis and debates. It is crucial to keep in mind that robots have a limited ability to see when they are situated on a specific vertex within the triangular grid $G$. The six nearby vertices that are directly linked to $v$ are the only vertices that these robots can see, and they are unable to see any vertices that are farther away.

**Time Cycle:** The operational framework of the robots follows a systematic and cyclical process known as the CHECK-COMPUTE-MOVE (CCM) cycle, which comprises three distinctive Rounds. These Rounds, namely CHECK, COMPUTE, and MOVE, are integral to the decision-making and movement behaviors of the robots within the system.

Each robot assesses its present node during the CHECK step and acquires pertinent data on both the node and its surrounding nodes. This process entails gathering information on a variety of variables, including the number of robots currently occupying the present node and the configuration of ports on neighbouring nodes. The robots build a thorough grasp of their immediate environment by careful observation of their surroundings, setting the foundation for later actions.

The robots move on to the COMPUTE step after the CHECK round, when they perform a deterministic algorithmic process. The information obtained in the preceding CHECK round is analysed and processed by the robots in this round. They may also converse with other robots connected to the same node in order to share pertinent information and ideas. The robots jointly deliberate, weigh their options, and arrive at well-informed judgements via this local computation. Depending on what is considered to be the best course of action, these choices may involve staying on the present node or choosing a particular port for leave.

The robots execute their judgements from the COMPUTE round in the last round, referred to as the MOVE round. A robot begins a movement by leaving the present node through the chosen port if it has identified the best port to exit through. Through this mobility, the robots are able to move around the system and switch between nodes based on their calculated decisions. The robots must move physically within the graph during the MOVE round in order to carry out the chosen tactics and advance towards the dispersion aim.

The iterative CHECK-COMPUTE-MOVE cycle, which is what distinguishes it, is crucial in enabling the robots' autonomous decision-making and coordinated behaviours. The robots continually assess their surroundings during the CHECK round, calculate local strategies during the COMPUTE round, and carry out the necessary motions during the MOVE round thanks to this cyclic process. The robots are constantly adapting and responding to dynamic changes in the system thanks to this iterative cycle, which enables efficient and effective dispersion. The robots continue a synchronised and coordinated effort to achieve the desired target of dispersion by continuously assessing, computing, and moving.

## 3.4 Scheduler

The Fully Synchronous (FSYNC) scheduler is one particular kind of scheduler that was the subject of this investigation. All robots are concurrently activated at the start of each of the equal-length global rounds that the FSYNC scheduler creates. All robots complete their calculations and moves in synchrony because to this synchronised activation.It is important to note that alternative scheduler types, such as the Semi-Synchronous (SSYNC) scheduler and the Asynchronous (ASYNC) scheduler, are frequently mentioned in the literature. Although the SSYNC scheduler similarly divides time into rounds, it allows for more flexibility in the activation patterns because not all robots are activated at the beginning of each round. The ASYNC scheduler, on the other hand, runs without a concept of global rounds and enables the independent activation of any robot at any time.The FSYNC and SSYNC schedulers offer valuable models for theoretical research, but because of their strict synchronisation requirements, they are viewed as less useful in real-world applications. However, investigating dispersion under the FSYNC scheduler sheds light on how dispersion algorithms behave and operate in synchronised activation situations.

In this research, we presupposed synchronous operation of the scheduler. The scheduler is a key component in determining when and how the robots are activated. As a result, the scheduler's activities are intended to test the robots and maybe impede their ability to achieve dispersion. In order to investigate the robustness and effectiveness of the dispersion algorithms under adversarial circumstances, where the robots must adjust to the scheduler's actions.

## 3.5 Problem Definition

The problem pertains to a triangular gird graph $G(l)$ consisting of $N$ nodes, where $l$ represents the number of layers. Within this graph, there exists a set of $k \leq n$ mobile robots that are initially positioned in an arbitrary manner on the nodes of $G$. The main objective of the Dispersion problem is to enable the robots to reposition themselves autonomously, ultimately achieving a configuration in which each robot occupies a distinct node within the graph $G$. In addressing this problem, the primary focus is on optimizing two key performance metrics: Time and Memory. Time refers to the number of rounds or steps required for the robots to accomplish the dispersion task, while Memory denotes the amount of bits that need to be stored at each individual robot for effective execution.

# CHAPTER 4
# Algorithm

We describe the algorithm $Disperse(k)$, where $k$ denotes the number of robots participating and $k$ is in order equivalent to the number of nodes $N$, to address the dispersion problem in triangular grid graphs. The algorithm, which was created expressly to speed up the dispersion process, runs through a sequential execution procedure with five unique steps.

The robots are first positioned on the border nodes of the triangular grid network during Round 1, which is the first phase of the algorithm. The robots are evenly dispersed along the graph's outer edges according to this calculating positioning.

In Round 2, the robots begin to advance from the boundary positions towards the triangular grid graph's four corner nodes. Consolidating the robots at these particular corner nodes will serve as a centralised starting point for the subsequent dispersion actions.

The algorithm attempts to assemble all $n$ robots in Round 3 at a preset corner node in the triangular grid graph, where n is the total number of robots. A vital stage in the dispersion process, this grouping of robots at a particular corner node prepares them for the upcoming activities.

The algorithm then shifts its attention to Round 4, when it focuses on dispersing the previously gathered robots onto the nodes of a single boundary of the triangular grid graph. The goal of this redistribution is to achieve a perfect dispersion pattern over the graph. The algorithm seeks to dispersion of the robots within the graph by carefully arranging the robots along a certain boundary.

The robots further disperse from a triangle grid graph boundary in the final round 5. In order to achieve the intended dispersion outcome, this dispersion aims to make sure that each node in the graph has exactly one robot. The algorithm ensures that each node is occupied by a single robot by carefully redistributing the robots from the boundary, achieving the problem's dispersion aim.

By following the sequential execution of these five rounds, Algorithm $Disperse(k)$

effectively tackles the Dispersion problem in triangular grid graphs. The algorithm's systematic approach guarantees the efficient and comprehensive dispersion of the k robots, ultimately resulting in a distributed configuration where each node of the graph is occupied by precisely one robot. Our algorithm solves rounds 1-5 in $O(O(max(H_G, W_G)))$ rounds.

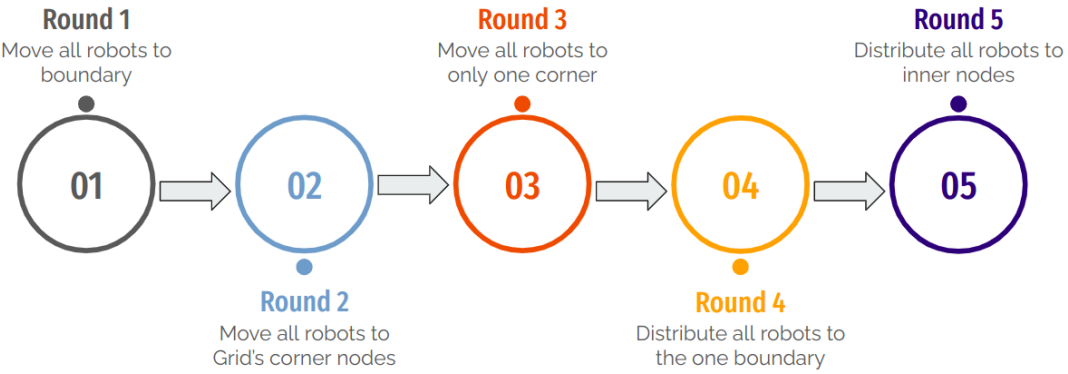


Figure 4.1: Rounds of the algorithm

Due to the robots' absence of a predetermined path to direct them towards the boundary nodes of the graph, Round 1's execution presents the first hurdle. The robots must use techniques to overcome this challenge, including methods for figuring out the best routes to take in order to successfully reach the boundary nodes.

In Rounds 2-4, the robots must only navigate and carry out tasks on the boundary nodes. This is the next challenge. This restriction restricts their movement and behaviours, necessitating the use of effective strategies to complete the needed objectives within this constrained environment.In Round 5, the third and final challenge appears. Its goal is to distribute the robots across the graph's nodes while making sure that each node is occupied by just one robot. To avoid numerous robots converging on the same node, this dispersion design requires precise coordination and synchronisation between the robots. The solution to this problem entails creating methods and algorithms that let the robots move around the network while avoiding collisions and making sure that each node is occupied by a single robot.

## 4.1 Functions and Variables

- $portCount(v)$: It will return the number of ports of that port or given node v.

- $isCorner(v)$ : It will check whether the node is a corner node.
- $portMap$ : It stores all ports of the node.
- $portEntered$ : It keeps the port where the robot entered the node.
- $L_i$ : Local leader robot Id chosen by all robots of the current node.
- $a_i$ : Local anchor robot Id chosen by all robots of the current node.
- $globalLeader$ : Global Leader Id of all robots.
- $currentLocalLeader$ : Local Leader Id of the current node.
- $idOfRank(N)$ : Returns the Id of an Nth number of the robot from the ascending list of robots' Ids.
- $settle()$ : Robot will settle on the current node and terminates the algorithm.

## 4.2 Round 1

**Algorithm 1:** Move_To_Boundary($r$)

```
   input  : r_i at arbitrary node
   output: r_i at boundary node
1  Choose random port w and exit through that port
2  while portCount(w) == 6 do
3      exit through port that clockwise 3rd port from the
4      port robots entered.
```

**Algorithm 2:** $isCorner(v)$

```
1  if portCount(v) == 2 then
2      return True
3  if portCount(v) == 3 then
4      for port_i ∈ portMap do
5          if portCount(port_i) == 4 then
6              return True
7  return False
```

The main goal of Round 1 is to move every robot to a boundary node in the triangular grid graph G. The robots choose a random port on their current node

and leave through that port in the first round. The goal of this operation is to initialise the variable *portEntered*, which will be used in Round 1's next phase.The robots on nodes with a degree of 6 will navigate towards the third port in relation to the port they entered through in the second round of Round 1. The robots go through this procedure repeatedly until they reach boundary nodes with degrees of 2, 3, 4, or 5. By continuing this process, the robots move strategically within the triangular grid graph, gradually approaching the required boundary nodes.

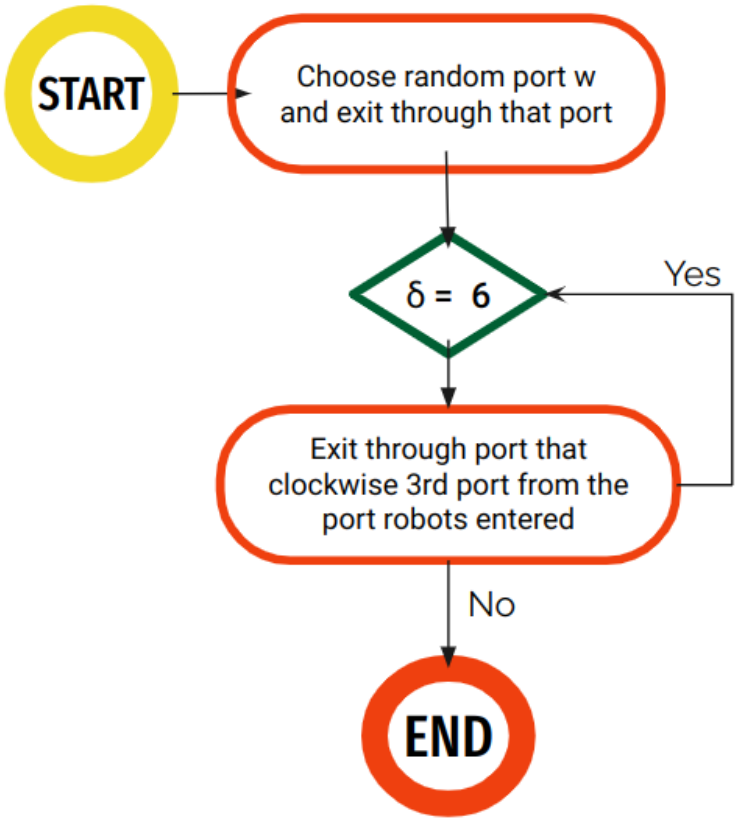


Figure 4.2: Flow of round 1

**Lemma 2.** *At the end of Round 1, all k robots will reach a boundary node of the Graph G with a time complexity of* $O(\max(H_G, W_G))$.

*Proof.* In start of the Round 1, For each robot $r \in R$ where $R$ is set of all robots. $r$ will choose random port $pu$ which takes robot to node $v$ from the node $u$. If node $v$ does not have exactly six ports, it means the robot has reached a boundary node and round ends for this robot. However, if node $v$ does have six ports, the robot needs to select a new port to exit through. The robot can move to an adjacent node $w$ by utilizing one of the available ports of node $v$, denoted as $pv_1, pv_2, pv_3, pv_4, pv_5, pv_6$. To determine the exit port, the robot $r$ remembers the

port $pu$ it used to enter node $v$ from $u$ and chooses the port that is three positions clockwise from $pu$. Let say this exit port is $pv$ and robot moves to the node $w$ using the passing edge $E$ which is between node $v$ and $w$. Let $\overrightarrow{E_{vw}}$ is the direction of $E$ starting from $v$ to $w$. This make $r$ to move same direction untill it reaches to a boundary node. The maximun possible distance travelled by robot r is diagonal of grid $O(\sqrt{N})$ which is $O(max(H_G, W_G))$.

□

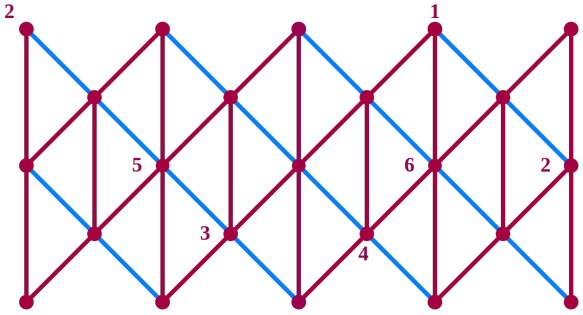


Figure 4.3: Initial Configuration of the triangular grid

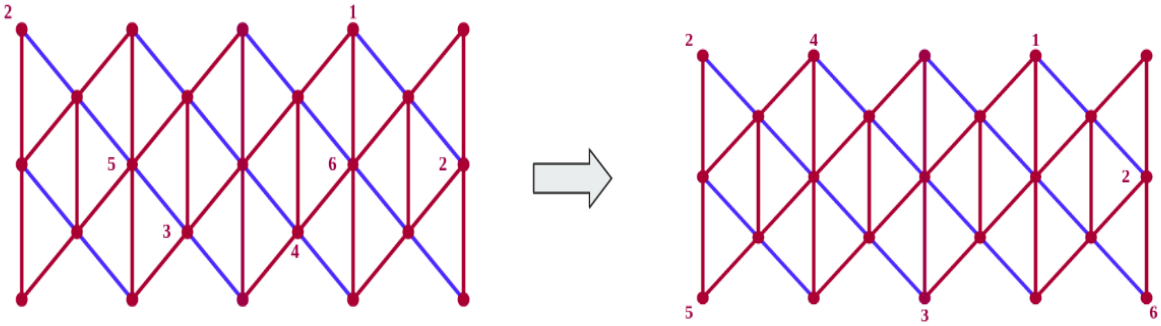


Figure 4.4: Transition of the robots to the boundary nodes in round 1

## 4.3 Round 2

The goal of Round 2 is to move every robot to one of the grid graph's four corner nodes. Based on the degree of the nodes that the robots are positioned on, the approach comprises unique movement rules for the robots. Details are as follows:

A robot that is situated on a node with a degree of 3 will travel to a node with a degree of 5 that is adjacent to it, omitting the port it entered through previously.

The robot keeps going through this process until it reaches a corner node. A robot will also go to an adjacent node with a degree of 5 if it is currently on a corner node with a degree of 3, except the entry port.A robot will only choose nodes with a degree of 4 as its next destination if it is currently on a node with a degree of 4, avoiding the port it used to enter the current node. Up until it encounters a corner node, the robot will go along degree-4 nodes. The robot will move to the adjacent node with a degree of 4, except the entry port, if it is currently on a corner node with a degree of 4.

**Algorithm 3:** Move_To_Any_Corner($r$)

```
   input  : r_i at boundary node v
   output: r_i at any corner node
 1 while isCorner() do
 2     if portCount(v) == 3 then
 3         for port_i ∈ portMap do
 4             if portCount(port_i) == 3 and port_i ≠ portEntered then
 5                 exit through port_i
 6             if portCount(port_i) == 5 and port_i ≠ portEntered then
 7                 exit through port_i
 8     if portCount(v) == 4 then
 9         for port_i ∈ portMap do
10             if portCount(port_i) == 2 or 3 then
11                 exit through port_i
12             if portCount(port_i) == 4 and port_i ≠ portEntered then
13                 exit through port_i
14     if portCount(v) == 5 then
15         for port_i ∈ portMap do
16             if portCount(port_i) == 2 then
17                 exit through port_i
18             if portCount(port_i) == 3 and port_i ≠ portEntered then
19                 exit through port_i
```

In fig 7, there is given result after Round 2 executes on $G(5)$ configuration.

**Lemma 3.** *At the end of Round 2, all k robots will be positioned at the boundary corner nodes of graph G with a time complexity of* $O(max(H_G, W_G))$.

*Proof.* The objective is to relocate all the robots to the boundary corner nodes that have a specific number of ports, namely 2 and 3. The goal is to move the robots in such a way that they reach these designated corner nodes.

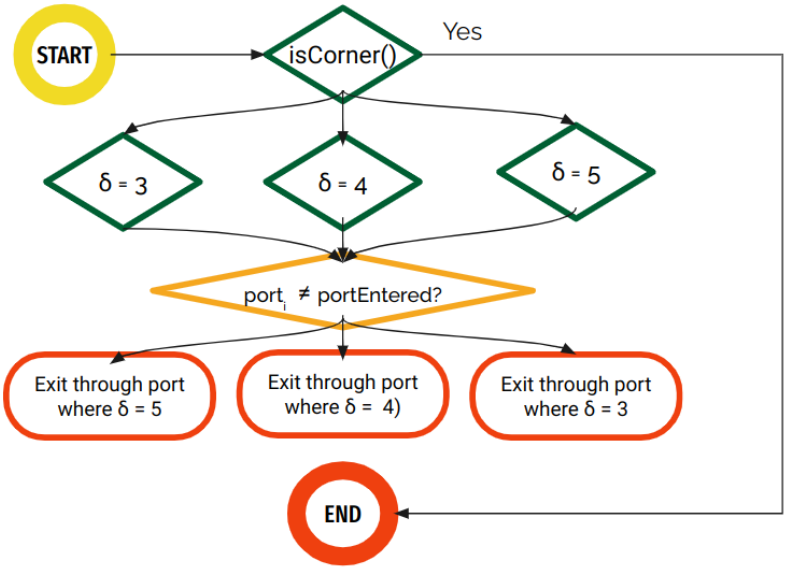


Figure 4.5: Flow of round 2

**Case 1:** When the robot $r_i$ is positioned at port 4 on a node, it will assess the port count of that node. If the port count is either 2 or 3, the robot $r_i$ will move towards the corresponding port that leads to a boundary corner node. On the other hand, if the port count is 4, the robot $r_i$ will move to the neighboring node through a port that is not the same as the one it entered from *portEntered*. It will then continue following the same process of moving towards the port opposite to the one it entered, until it eventually reaches a boundary corner node.The maximum distance robot will travel in this Round is $O(\max(H_G, W_G))$.
**Case 2:** When the robot $r_i$ is on port 3 or 5. The three nodes *portCount* is $p1, p2, p3$. If any port counts are 2 or 3, then the $r_i$ robot moves to that node as a boundary corner node. If not, then he will choose the port whose count is 5 because if he chooses a port with a count of 6 then we have to rerun Round 1, and at the end, it may be only the boundary node again. Similarly, for a port whose count is 5, either he will choose a port whose count is 2 or 3 or either port whose port count is 3, and following the same procedure, all the robots at the end of this Round will reach to boundary corner node. The time complexity of this is $O(max(H_G, W_G))$. □

## 4.4 Round 3

In this Round, all robots at the corners of the grid need to be gathered at a single corner. To accomplish this, each corner has a local leader, identified as the robot with the minimum ID. The local leaders move towards the intersection of the

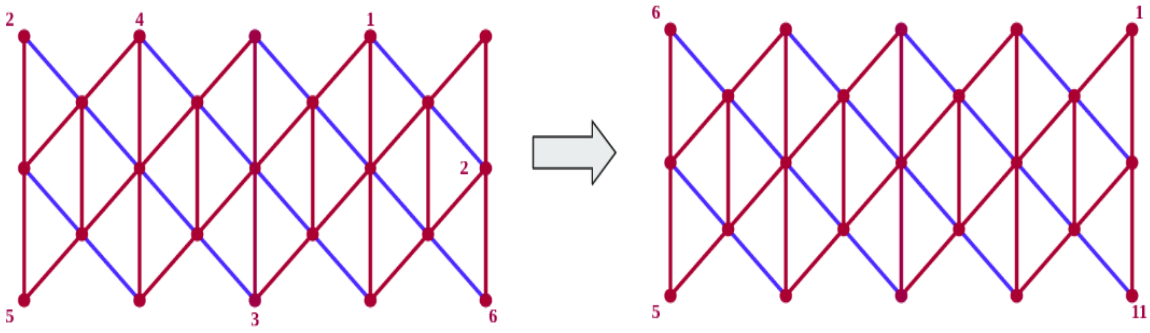


Figure 4.6: Transition of the robots to the corner nodes in round 2

boundaries to find the global leader. Once all robots see the global leader, they move towards it and stop upon reaching it.

**Algorithm 4:** Move_To_One_Corner($r$)

**input :** $r_i$ at any corner node $v$
**output:** $r_i$ at corner node which have robot with smallest ID

Choose a local leader $L_i = \min(\forall id \, \epsilon \, S_{localId})$
Set an anchor robot $a_i = \max(\forall id \, \epsilon \, S_{localId})$
**if** $L_i == ID$ **then**
    $leader = True$
**if** $leader == True$ ***and*** $robotCount \geq 2$ **then**
    **for** $port_i \in portMap$ **do**
        **if** $portCount(port_i) == 4$ **then**
            exit through $port_i$
**while** $leader == True$ ***and*** $a_i \neq a_{current}$ **do**
    $globalLeader = \min(globalLeader, currentLocalLeader)$
    run $Move\_To\_Any\_Corner(r)$
**while** $currentLocalLeader \neq globalLeader$ **do**
    run $Move\_To\_Any\_Corner(r)$

**Lemma 4.** *At the end of Round 3, all k robots will be positioned on a single corner node. The time complexity of Round 3 can be described as* $O(max(H_G, W_G))$ .

*Proof.* First, we will choose the local leader that is the minimum ID at that port from all boundary corner nodes $a_i$; we move the robots by using the idea similar to Round 2 to other boundary corner nodes to choose a global leader. For that, we will use leader election in a ring network where each node with a local leader sends its ID around the grid using Round 2, and a node forwards a received ID

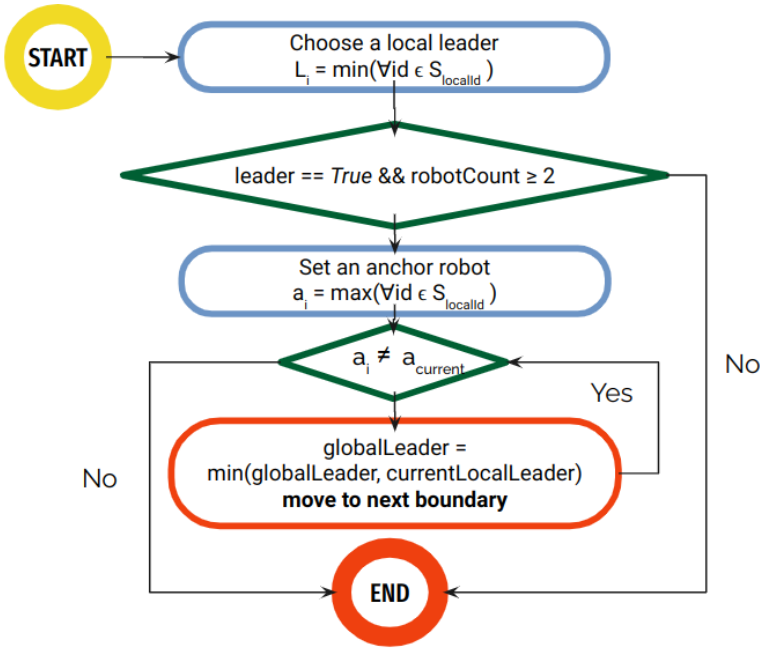


Figure 4.7: Flow of round 3

to its neighbor only if the received ID is smaller in value compared to its ID. If a node gets its ID, it elects itself a leader and broadcasts the message to other nodes, and then another node moves to that node using Round 2. The longest distance that a robot can travel is $4 * (O(\max(H_G, W_G)))$. □

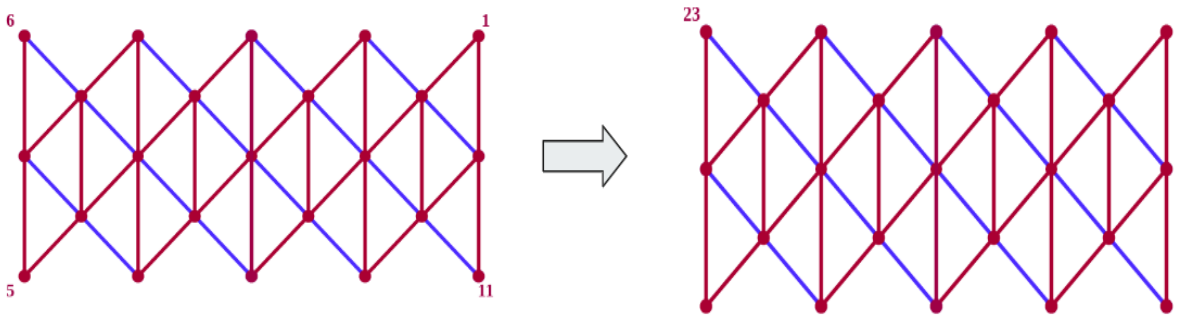


Figure 4.8: Transition of the robots to any one corner in round 3

## 4.5 Round 4

In this Round, all the robots are already on one of the four corners of the Grid. This Round aims to distribute robots on the adjacent boundary of the corner node

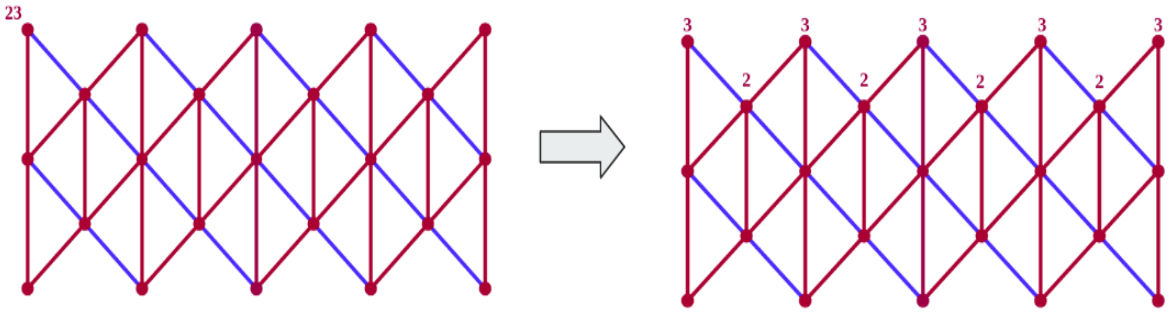


Figure 4.9: Distribution of the robots to the adjacent boundary nodes in round 4

where all robots are. At the corner, the node has two adjacent nodes, one with 4 degrees and one with a 5-degree node. In our algorithm, robots are distributed over the boundary, which has degrees 3 and 5 nodes. At the start of this Round, there are k robots, which are there at the node. Now there are two types of node columns: one with $p = ceil(L/2)$ number of nodes and the other with $q = L - p$ number of nodes. So alternatively, we will distribute $p$ and $q$ robots according to the columns of the Grid. In fig 13, there is given result after Round 4 executes on $G(5)$ configuration.

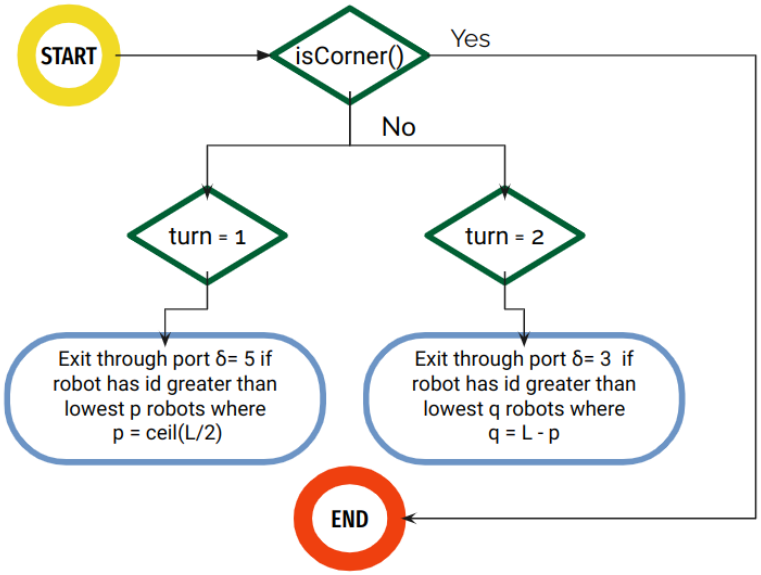


Figure 4.10: Flow of round 4

**Lemma 5.** *At the end of Round 4, all k robots are distributed along the boundary of G in such a way that each node on the boundary contains precisely either p or q robots, where p and q are values that satisfy the equation $p + q = l$, where l is number of the layers.*

**Algorithm 5:** Distribute_Over_Boundary($r$)

**input :** $r_i$ at corner node $v$
**output:** $r_i$ at boundary node

```
1  p = ceil(L/2) /* Number of nodes in odd vertical line          */
2  q = L − p /* Number of nodes in even vertical line            */
3  if portCount(v) == 3 then
4      swap(p, q)
5  turn = 1
6  N =Number of robots at current node
7  lastId = idOfRank(N − p)
8  if id > lastId then
9      for port_i ∈ portMap do
10         if portCount(port_i) == 3 or 5 then
11             exit through port_i
12 turn = 2
13 while isCorner() do
14     if turn == 1 then
15         lastId = idOfRank(N − p)
16         if id > lastId then
17             for port_i ∈ portMap do
18                 if portCount(port_i) == 5 then
19                     exit through port_i
20         turn = 2
21     if turn == 2 then
22         lastId = idOfRank(N − q)
23         if id > lastId then
24             for port_i ∈ portMap do
25                 if portCount(port_i) == 3 then
26                     exit through port_i
27         turn = 1
```

*Proof.* All robots at the corner node will exit through ports 3 or 5, whichever is available in the configuration, except the lowest p robots. So all robots will choose the same direction as there is only one possible way. On each node, p or q robots will stay on the node, and the rest of $N - (p \text{ or } q)$ will leave the node. So till the corner node, all robots will be distributed on the nodes. For $config(2, 2, 2, 2)$, Total robots distributed $= p * (L) + q(L - 1) = N$ Similarly, we can prove this for other configurations. The maximum distance robot will travel is $O(W_G)$. □

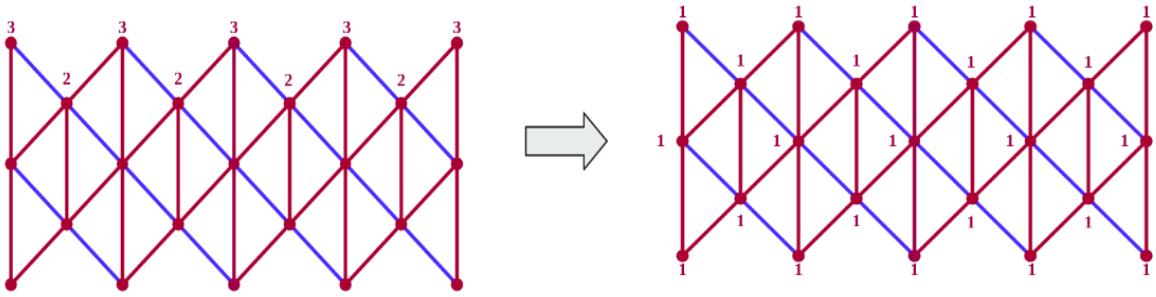

Figure 4.11: Distribution of the robots to the inner nodes in round 5

## 4.6 Round 5

In the final round, robots are placed along one of the four boundaries of a given configuration. The objective is to distribute all the robots from the boundary to the inner nodes. The given figure 27 illustrates the outcome after executing Round 2 on a configuration represented by graph G(5).

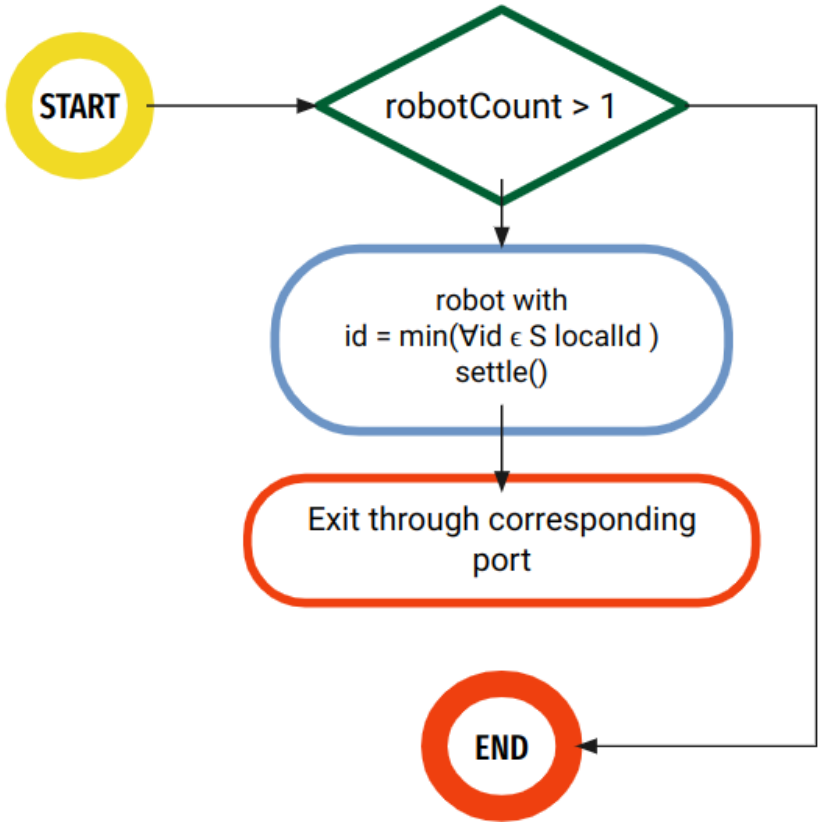


Figure 4.12: Flow of round 5

**Algorithm 6:** Distribute_Over_Grid($r$)

**input :** $r_i$ at boundary node $v$
**output:** $r_i$ at node where $robotCount = 1$

```
if isCorner() then
    while robotCount(v) > 1 do
        if id == min(∀id ϵ S_localId) then
            settle() break
        for port_i ∈ portMap do
            if portCount(port_i) == 4 then
                exit through port_i
    settle()
    break
if portCount(v) == 3 then
    if id == min(∀id ϵ S_localId) then
        settle() break
    for port_i ∈ portMap do
        if portCount(port_i) == 6 then
            exit through port_i
if portCount(v) == 5 then
    if id == min(∀id ϵ S_localId) then
        settle() break
    port_a = 2nd port clockwise from portEntered
    port_b = 2nd port countrer clockwise from
    port_x such that portCount(port_x) == 3 and
    port_x ≠ portEntered
    if port_a == port_b then
        exit through port_a
    else
        exit through 3rd port counter clockwise from portEntered
if portCount(v) == 6 then
    while robotCount(v) > 1 do
        if id == min(∀id ϵ S_localId) then
            settle() break
        exit through port that clockwise 3rd port from the port robots
         entered.
    settle()
    break
```

**Lemma 6.** *At the conclusion of Round 5, in configuration G with $k = \Omega(n)$ robots, the distribution of robots over the triangular grid G ensures that each node contains at most*

*one robot.*

*Proof.* In the case of corner nodes, the nodes' robots will move to the node with degree 4, and the robot with minimum id will settle on the node in each iteration. In the case of degree 3 and 5 nodes, they need to find the correct port to move the rest robots. So they will check $port_a$ and $port_b$, which are found from adjacent ports of the boundary nodes. Using this method, we can see the next port to move, and robots can distribute vertically. The maximum distance robot will travel $H_G$, so the upper bound for Round 5 is $O(H_G)$. □

**Theorem 1.** *The $Disperse(k)$ algorithm can effectively solve the dispersion problem on a triangular grid within a time complexity of $O(\max(H_G, W_G))$ rounds while utilizing only $O(\log k)$ bits for the mobile robots. Here, $H_G$ represents the height of the grid and $W_G$ denotes the width of the grid.*

*Proof.* To establish the correctness of the $Disperse(k)$ algorithm, we combine the correctness proofs for Lemmas 1 through 6. Each lemma contributes to ensuring the correctness of the algorithm in solving the dispersion problem on the triangular grid. The total time requires for running the whole algorithm is $O(max(H_G, W_G))$ as each round takes $O(max(H_G, W_G))$. Regarding memory bits, variables $portEntered$, $portExited$, $turn$, $portCount$, $portMap$ will take $O(1)$ bits as $\Delta = 6$ and $id$, $localLeader$, $globalLeader$ will take $O(\log k)$ bits. □

# CHAPTER 5
# Conclusions and Future Work

This Dispersion problem on a Triangular Grid holds great importance due to its close connection to various fundamental robot coordination problems. Through the introduction of a specific set of configurations, we addressed this problem and presented an algorithm consisting of five Rounds. Notably, our algorithm achieves an optimal solution in terms of both time complexity, requiring only $O(\max(H_G, W_G))$ rounds, and memory usage, utilizing merely $O(\log k)$ bits for the mobile robots. This result establishes an optimal solution for the Dispersion problem on a given configuration of a triangular grid, satisfying the constraints of time efficiency and memory optimization.

It indeed holds significant interest to explore further avenues related to the Dispersion problem. One intriguing direction involves proving a lower bound of $\Omega(k)$ in terms of time complexity, or alternatively, devising an algorithm with a time complexity of $O(\sqrt{k})$ for configurations where $k < \Omega(N)$ on both square and triangular grid graphs. Additionally, extending our algorithms to address Dispersion in semi-synchronous and asynchronous settings presents another promising avenue for research.

Furthermore, considering the Dispersion problem on arbitrary graphs, it would be compelling to develop an algorithm that can solve it within a time complexity of $O(k)$ or improve the existing time lower bound from $\Omega(k)$ to $\Omega(\min(m, k))$, where $m$ represents the number of edges in the graph. These directions open up new possibilities for advancing our understanding of the Dispersion problem and its applicability to various graph structures and computational models.

# References


[1] J. Augustine and W. K. Moses Jr. Dispersion of mobile robots: A study of memory-time trade-offs. *CoRR*, abs/1707.05629, 2018. A preliminary version appeared in ICDCN'18.

[2] E. Bampas, L. Gąsieniec, N. Hanusse, D. Ilcinkas, R. Klasing, and A. Kosowski. Euler tour lock-in problem in the rotor-router model. In I. Keidar, editor, *Distributed Computing*, pages 423–435, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[3] L. Barrière, P. Flocchini, and E. Barrameda. Uniform scattering of autonomous mobile robots in a grid. *Int. J. Found. Comput. Sci.*, 22:679–697, 01 2011.

[4] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, August 2008.

[5] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, October 1989.

[6] G. D'Angelo, G. Di Stefano, R. Klasing, and A. Navarra. Gathering of robots on anonymous grids and trees without multiplicity detection. *Theoretical Computer Science*, 610:158–168, 2016. Structural Information and Communication Complexity.

[7] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609:171–184, 2016.

[8] D. Dereniowski, Y. Disser, A. Kosowski, D. Pajak, and P. Uznański. Fast collaborative graph exploration. *Inf. Comput.*, 243(C):37–49, August 2015.

[9] Y. Elor and A. M. Bruckstein. Uniform multi-agent deployment on a ring. *Theor. Comput. Sci.*, 412(8-10):783–795, 2011.

[10] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[11] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Mobile Entities*, volume 1 of *Theoretical Computer Science and General Issues*. Springer International Publishing, 2019.

[12] P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.

[13] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, November 2005.

[14] P. Goswami, A. Sharma, S. Ghosh, and B. Sau. Time optimal gathering of myopic robots on an infinite triangular grid. In S. Devismes, F. Petit, K. Altisen, G. A. Di Luna, and A. Fernandez Anta, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 270–284, Cham, 2022. Springer International Publishing.

[15] T.-R. Hsiang, E. M. Arkin, M. A. Bender, S. Fekete, and J. S. B. Mitchell. Online dispersion algorithms for swarms of robots. In *SoCG*, pages 382–383, 2003.

[16] T.-R. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *WAFR*, pages 77–94, 2002.

[17] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. *Theoretical Computer Science*, 390(1):27–39, 2008.

[18] A. D. Kshemkalyani and F. Ali. Efficient dispersion of mobile robots on graphs. In *ICDCN*, pages 218–227, 2019.

[19] A. Menc, D. Pajak, and P. Uznanski. Time and space optimality of rotor-router graph exploration. *Inf. Process. Lett.*, 127:17–20, 2017.

[20] P. Poudel and G. Sharma. Time-optimal uniform scattering in a grid. In *ICDCN*, pages 228–237, 2019.

[21] M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, and T. Masuzawa. Uniform deployment of mobile agents in asynchronous rings. In *PODC*, pages 415–424, 2016.

[22] R. Subramanian and I. D. Scherson. An analysis of diffusive load-balancing. In *SPAA*, pages 220–225, 1994.

# CHAPTER A
# Implementation in C++

```
/**
 *    author:  Yash Joshi
**/

#include <bits/stdc++.h>

using namespace std;


struct Node {
    int x;
    int y;

    Node() {
        x = 0;
        y = 0;
    }

    Node(int xx, int yy) {
        x = xx;
        y = yy;
    }
};

int leaderOfNode(Node node);

int anchorOfNode(Node node);

void broadcastGlobalLeader(int leader);

int layers = 0;

class Grid {
    int rows;
```

```
    int cols;

public:
    vector<vector<int>> cells;
    vector<vector<int>> robotCount;

    Grid(int rows, int cols) {
        this->rows = rows;
        this->cols = cols;

        cells.resize(rows);
        for (int i = 0; i < rows; i++)
            cells[i].resize(cols, 0);

        robotCount.resize(rows);
        for (int i = 0; i < rows; i++)
            robotCount[i].resize(cols, 0);
    }

    /* Checking the cordinates are valid or not */
    bool isValid(Node node) {
        int i = node.x, j = node.y;
        if (i < 0 || j < 0 || i >= rows || j >= cols)
            return false;
        else return true;
    }

    /* Number of Ports for the given Node */
    int portCount(Node node) {
        vector<Node> portMap = getPortMap(node);
        return portMap.size();
    }

    vector<Node> getPortMap(Node node) {
        vector<Node> portMap;

        /* 6 possible directions for node */
        vector<vector<int>> dirs = {
                {-2, 0},
                {-1, 1},
                {1,  1},
                {2,  0},
                {1,  -1},
                {-1, -1},
        };
```

```
    for (auto dir: dirs) {
        Node tempNode(node.x + dir[0], node.y + dir[1]);
        if (isValid(tempNode)) {
            portMap.push_back(tempNode);
        }
    }

    return portMap;
}

/* Checking node is corner node or not*/
bool isCorner(Node node) {
    if (portCount(node) == 2) return true;
    else if (portCount(node) == 3) {
        vector<Node> portMap = getPortMap(node);
        for (Node i: portMap) {
            if (portCount(i) == 2 || portCount(i) == 4)
                return true;
        }
    }
    return false;
}

void setRobotCount(Node node, int val) {
    robotCount[node.x][node.y] = val;
}


int getAnchor(Node node) {


    return 0;
}

int getLeader(Node node) {

    return leaderOfNode(node);
}

int getRobotCount(Node node) {
    return robotCount[node.x][node.y];
}

~Grid() {}
```

```
};
```

```
6 9
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
8
3 5
1 1
0 2
3 1
4 8
4 8
2 2
2 2
```

Figure A.1: Sample input file

```
class Robot {
private:
    int robotID;

    int portEntered = 0;
    Grid *grid;
    bool leader = false;
    bool anchor = false;


public:
    int globalLeader = -1;
    Node currNode;
    Node prevNode;
```

```
Robot() {}

void setPosition(int x, int y) {
    currNode.x = x;
    currNode.y = y;
}

/* Set pointer to grid to use Grid methods */
void setGrid(Grid *pGrid) {
    this->grid = pGrid;
}

void setId(int id) {
    this->robotID = id;
}

bool isSameNode(Node a, Node b) {
    return (a.x == b.x) && (a.y == b.y);
}

void setPortEntered(vector<Node> &portMap, Node node) {
    for (int index = 0; index < portMap.size(); index++) {
        if (isSameNode(prevNode, portMap[index])) {
            portEntered = index;
            return;
        }
    }
}

void printPos(Node node) {
    cout << node.x << " : " << node.y << endl;
}

void exitToPort(vector<Node> &portMap, Node node) {
    prevNode = currNode;
    currNode = node;
    portMap = grid->getPortMap(currNode);

    grid->setRobotCount(prevNode,
    grid->getRobotCount(prevNode) - 1);
    grid->setRobotCount(currNode,
    grid->getRobotCount(currNode) + 1);

}
```

```
/* Round 1*/
void moveToBoundary() {
    /* Choose random port and exit through that port*/
    int random = 1;
    vector<Node> portMap = grid->getPortMap(currNode);

    exitToPort(portMap,
    portMap[(random) % portMap.size()]);

    setPortEntered(portMap, prevNode);
    /* Exit through 3rd Node from portEnterd */
    while (grid->portCount(currNode) == 6) {
        exitToPort(portMap,
        portMap[(portEntered + 3) % portMap.size()]);
        setPortEntered(portMap, prevNode);
    }

    /* Print Position */

}
```

```
Stage 1 starts...
Robot 0 Position : (3,5) → (1,7)
Robot 1 Position : (1,1) → (4,4)
Robot 2 Position : (0,2) → (4,2)
Robot 3 Position : (3,1) → (1,3)
Robot 4 Position : (4,8) → (5,7)
Robot 5 Position : (4,8) → (5,7)
Robot 6 Position : (2,2) → (1,3)
Robot 7 Position : (2,2) → (1,3)
```

Figure A.2: Output of round 1

```
/* Round 2*/
void moveToAnyCorner() {
    while (!grid->isCorner(currNode)) {
        vector<Node> portMap = grid->getPortMap(currNode);
        if (grid->portCount(currNode) == 3) {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 3) ||
```

```
                (grid->portCount(node) == 5))
                    && (!isSameNode(node, portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap, prevNode);
                    break;
                }
            }
        }
        if (grid->portCount(currNode) == 4) {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 2) ||

                (grid->portCount(node) == 3) ||
                (grid->portCount(node) == 4))
                    && (!isSameNode(node, portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap, prevNode);
                    break;
                }
            }
        }
        if (grid->portCount(currNode) == 5) {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 2) ||
                (grid->portCount(node) == 3))
                    && (!isSameNode(node, portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap, prevNode);
                    break;
                }
            }
        }
    }
}
```

```
/* Round 3*/
void moveToOneCorner() {
    int myLeader = leaderOfNode(currNode);

    if (myLeader == robotID)
        leader = true;

    int myAnchor = anchorOfNode(currNode);

    int currAnchor = -1;
    int currLeader = myLeader;
```

```
Stage 2 starts...
Robot 0 Position : (1,7) → (0,8)
Robot 1 Position : (4,4) → (4,8)
Robot 2 Position : (4,2) → (4,8)
Robot 3 Position : (1,3) → (0,8)
Robot 4 Position : (5,7) → (4,0)
Robot 5 Position : (5,7) → (4,0)
Robot 6 Position : (1,3) → (0,8)
Robot 7 Position : (1,3) → (0,8)
```

Figure A.3: Output of round 2

```
if (myAnchor == robotID)
    anchor = true;

if (anchor && leader)
    leader = false;


if (leader) {
    int tempLeader = myLeader;
    while (currAnchor != myAnchor) {
        //go to intial first node near to current corner
        vector<Node> portMap = grid->getPortMap(currNode);
        for (Node node: portMap) {
            if (((grid->portCount(node) == 3) ||
            (grid->portCount(node) == 5) ||
            (grid->portCount(node) == 4))
                && (!isSameNode(node, portMap[portEntered]))) {
                exitToPort(portMap, node);
                setPortEntered(portMap, prevNode);
                break;
            }
        }
        moveToAnyCorner();
        tempLeader = min(tempLeader, leaderOfNode(currNode));
        currAnchor = anchorOfNode(currNode);
    }
```

```
            broadcastGlobalLeader(tempLeader);
        }


        currLeader = leaderOfNode(currNode);

        while (currLeader != globalLeader) {
            //go to intial first node near to current corner
            vector<Node> portMap = grid->getPortMap(currNode);
            for (Node node: portMap) {
                if (((grid->portCount(node) == 3) ||
                (grid->portCount(node) == 5) ||
                (grid->portCount(node) == 4))
                    && (!isSameNode(node, portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap, prevNode);
                    break;
                }
            }

            moveToAnyCorner();
            currLeader = leaderOfNode(currNode);
        }


    }
```

```
Stage 3 starts...
Robot 0 Position : (0,8) → (0,8)
Robot 1 Position : (4,8) → (0,8)
Robot 2 Position : (4,8) → (0,8)
Robot 3 Position : (0,8) → (0,8)
Robot 4 Position : (4,0) → (0,8)
Robot 5 Position : (4,0) → (0,8)
Robot 6 Position : (0,8) → (0,8)
Robot 7 Position : (0,8) → (0,8)
```

Figure A.4: Output of round 3

```
/* Round 4*/
void distributeOverBoundary() {
    int p = ceil(layers / 2);
    int q = layers - p;

    if (grid->portCount(currNode) == 3)
        swap(p, q);

    bool turn = 1;

    int robotCount = grid->getRobotCount(currNode);

    int currentLot = p;
    vector<Node> portMap = grid->getPortMap(currNode);

    if (robotID >= currentLot) {
        for (Node node: portMap) {
            if (((grid->portCount(node) == 3) ||
            (grid->portCount(node) == 5))) {
                exitToPort(portMap, node);
                setPortEntered(portMap, prevNode);
                break;
            }
        }
    }
    currentLot += q;

    while (robotID >= currentLot) {
        if (turn) {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 3)
                || (grid->portCount(node) == 5))
                    && (!isSameNode(node,
                    portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap,
                    prevNode);
                    break;
                }
            }
            currentLot += p;
        } else {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 3)
                || (grid->portCount(node) == 5))
```

```
                    && (!isSameNode(node,
                    portMap[portEntered])) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap,
                    prevNode);
                    break;
                }
            }
            currentLot += q;
        }
        turn = !turn;
    }
}
```

```
Stage 4 starts...
Robot 0 Position : (0,8) → (0,8)
Robot 1 Position : (0,8) → (0,8)
Robot 2 Position : (0,8) → (0,8)
Robot 3 Position : (0,8) → (1,7)
Robot 4 Position : (0,8) → (1,7)
Robot 5 Position : (0,8) → (1,7)
Robot 6 Position : (0,8) → (0,6)
Robot 7 Position : (0,8) → (0,6)
```

Figure A.5: Output of round 4

```
/* Round 5*/
void distributeOverGrid() {
    while (grid->getRobotCount(currNode) != 0) {
        vector<Node> portMap =
        grid->getPortMap(currNode);

        if (leaderOfNode(currNode) == robotID) {
            cout << robotID << " settled on : ";
            printPos(currNode);
            return;
        }
        if ((grid->portCount(currNode) == 2) ||
            ((grid->portCount(currNode) == 3) &&
            (grid->isCorner(currNode))) ||
```

```
            (grid->portCount(currNode) == 4)) {
            for (Node node: portMap) {
                if (((grid->portCount(node) == 2)
                || (grid->portCount(node) == 3) ||
                (grid->portCount(node) == 4))
                    && ((!isSameNode(node,
                    portMap[portEntered])) ||
                    (grid->isCorner(currNode)))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap,
                    prevNode);
                    break;
                }
            }
            continue;
        } else if (grid->portCount(currNode) == 3) {
            for (Node node: portMap) {
                if ((grid->portCount(node) == 6)
                    && (!isSameNode(node,
                    portMap[portEntered]))) {
                    exitToPort(portMap, node);
                    setPortEntered(portMap,
                    prevNode);
                    break;
                }
            }
            continue;
        } else if (grid->portCount(currNode) == 5) {
            bool hasCorner = false;
            for (Node node: portMap) {
                if (grid->isCorner(node)) {
                    hasCorner = true;
                    break;
                }
            }

            if (hasCorner) {
                // 6 degree node after 4 degree node
                for (int i = 0; i < portMap.size(); i++) {
                    if (grid->portCount(portMap[i]) == 4) {
                        Node prev, next;
                        if (i == 0) {
                            prev = portMap[portMap.size() - 1];
                        } else prev = portMap[i - 1];
```

```
                    if (i == portMap.size() - 1) {
                        next = portMap[0];
                    } else next = portMap[i + 1];

                    if (grid->portCount(prev) == 6) {
                        exitToPort(portMap, prev);
                        setPortEntered(portMap, prevNode);
                    } else if (grid->portCount(next) == 6) {
                        exitToPort(portMap, next);
                        setPortEntered(portMap, prevNode);
                    }
                }
            }

        } else {
            //2nd 6 degree node
            bool firstTime = true;
            Node prev = portMap[portMap.size() - 1], next;
            for (int i = 0; i < portMap.size(); i++) {
                if (i == portMap.size() - 1) {
                    next = portMap[0];
                } else next = portMap[i + 1];

                if ((grid->portCount(portMap[i]) == 6) &&
                    (grid->portCount(prev) == 6) &&
                    (grid->portCount(next) == 6)
                        ) {
                    exitToPort(portMap, portMap[i]);
                    setPortEntered(portMap, prevNode);
                }
                prev = portMap[i];
            }
        }


        continue;
    } else if (grid->portCount(currNode) == 6) {
        exitToPort(portMap,
        portMap[(portEntered + 3) %
        portMap.size()]);
        setPortEntered(portMap, prevNode);
        continue;
    }
}
```

```
    }

    int getId() {
        return robotID;
    }


    void setGlobalLeader(int x) {
        this->globalLeader = x;
    }

    ~Robot() {}
};
```

```
Stage 5 starts...
Robot 0 Position : (0,8) → (0,8)
Robot 1 Position : (0,8) → (2,8)
Robot 2 Position : (0,8) → (4,8)
Robot 3 Position : (1,7) → (1,7)
Robot 4 Position : (1,7) → (3,7)
Robot 5 Position : (1,7) → (5,7)
Robot 6 Position : (0,6) → (0,6)
Robot 7 Position : (0,6) → (2,6)
```

Figure A.6: Output of round 5

```
void printRobotsPos();

void initIO() {
    char inputFile[] = {"input.txt"},
    outputFile[] = {"output.txt"};
    freopen(inputFile, "r", stdin);
    freopen(outputFile, "w", stdout);
}

Robot robots[10];
int noOfRobots;

int main() {
    initIO();
```

```
/* Take Configuration */
int rows, cols;
cin >> rows >> cols;
layers = rows;

Grid grid(rows, cols);
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cin >> grid.cells[i][j];
    }
}


cin >> noOfRobots;


int xx, yy;

for (int i = 0; i < noOfRobots; i++) {
    cin >> xx >> yy;
    robots[i].setPosition(xx, yy);
    robots[i].setGrid(&grid);
    robots[i].setId(i);
}

/* Round 1 */
for (int i = 0; i < noOfRobots; i++)
robots[i].moveToBoundary();

/* Round 2 */
for (int i = 0; i < noOfRobots; i++)
robots[i].moveToAnyCorner();

/* Round 3 */
for (int i = 0; i < noOfRobots; i++)
robots[i].moveToOneCorner();

/* Round 4 */
for (int i = 0; i < noOfRobots; i++)
robots[i].distributeOverBoundary();

/* Round 5 */
for (int i = 0; i < noOfRobots; i++)
robots[i].distributeOverGrid();
```

```
    /* Final Positions of Robots */
    return 0;
}


int leaderOfNode(Node node) {
    int leaderId = INT_MAX;
    for (int i = 0; i < noOfRobots; i++) {
        if ((robots[i].currNode.x == node.x) &&
        (robots[i].currNode.y == node.y)) {
            leaderId = min(robots[i].getId(), leaderId);
        }
    }
    return leaderId;
}

int anchorOfNode(Node node) {
    int leaderId = INT_MIN;
    for (int i = 0; i < noOfRobots; i++) {
        if ((robots[i].currNode.x == node.x) &&
        (robots[i].currNode.y == node.y)) {
            leaderId = max(robots[i].getId(), leaderId);
        }
    }
    return leaderId;
}

void broadcastGlobalLeader(int leader) {
    for (int i = 0; i < noOfRobots; i++) {
        if (robots[i].globalLeader == -1)
            robots[i].globalLeader = leader;
        else break;
    }
}
```