Practical Considerations and Applications for Autonomous Robot Swarms

Rory Alan Hector

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

Part of the Artificial Intelligence and Robotics Commons, Electrical and Computer Engineering Commons, Robotics Commons, and the Theory and Algorithms Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_dissertations/5809

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
PRACTICAL CONSIDERATIONS AND APPLICATIONS FOR AUTONOMOUS ROBOT SWARMS

A Dissertation
Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College in partial fulfillment of the requirements for the degree of Doctor of Philosophy
in
The Division of Electrical and Computer Engineering

by
Rory Alan Hector
B.S., Louisiana State University, 2018
May 2022
This dissertation is dedicated to Steven P. Hector.
Acknowledgments

First, I would like to thank my advisor, Dr. Vaidyanathan for all of his advice and support over the years. He has always gotten me to see things from a different perspective, and has been an exceptional teacher.

I also would like to thank the members of my committee — Dr. Trahan, Dr. Wei, Dr. Nguyen, Dr. Busch, and Dr. Peng — who have offered their time to provide valuable feedback along the way.

Next, I couldn’t have done this without the assistance of the Department of Defense Science, Mathematics, and Research for Transformation (SMART) Scholarship-for-Service Program. I am also grateful to Mr. Joseph F. Domino for his donations to the College of Engineering in order to provide the Superior Graduate Student Scholarship.

Finally, I’d like to thank my wonderful partner for all of her support over the years and my family for always keeping me motivated.
# Table of Contents

Acknowledgments ................................................................ iv
List of Tables ........................................................................ vii
List of Figures ......................................................................... viii
Symbols and Acronyms ............................................................ x
Abstract ................................................................................. xi

Chapter 1. Introduction .......................................................... 1
  1.1. Contributions ............................................................... 2
  1.2. Dissertation Roadmap .................................................. 3

Chapter 2. Robot Swarm Models ............................................. 4
  2.1. Base Robot Model ....................................................... 4
  2.2. Activation Schedules ................................................... 5
  2.3. Memory .................................................................... 7
  2.4. Visibility .................................................................. 8
  2.5. Extent ..................................................................... 8
  2.6. Performance Measures .............................................. 9
  2.7. Model Selection ........................................................ 11

Chapter 3. Literature Review ................................................. 12

Chapter 4. Distance and Spatial Complexity ........................... 16
  4.1. Contributions of This Chapter ................................... 17
  4.2. Chapter Roadmap ...................................................... 17
  4.3. Convex Polygon Based Complete Visibility .................... 18
  4.4. Distance Complexity ............................................... 19
  4.5. Distance Analysis .................................................... 22
  4.6. Spatial Complexity .................................................. 33
  4.7. Other Applications .................................................... 35
  4.8. Conclusion ............................................................... 36

Chapter 5. Convex Polygon Based Complete Visibility on a Grid 37
  5.1. Contributions of This Chapter ................................... 39
  5.2. Previous Work .......................................................... 40
  5.3. Chapter Roadmap ...................................................... 41
  5.4. Preliminaries ............................................................. 42
  5.5. Lower Bounds .......................................................... 43
  5.6. Coprimes Strategy .................................................... 45
  5.7. Coprimes Algorithm Framework ................................. 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8. Quadratic Algorithm</td>
<td>59</td>
</tr>
<tr>
<td>5.9. Ternary Tree Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>5.10. Farey Sequence Algorithm</td>
<td>69</td>
</tr>
<tr>
<td>5.11. Conclusion</td>
<td>70</td>
</tr>
<tr>
<td>Chapter 6. Doorway Egress by Point Robots</td>
<td>72</td>
</tr>
<tr>
<td>6.1. Contributions of This Chapter</td>
<td>72</td>
</tr>
<tr>
<td>6.2. Previous Work</td>
<td>73</td>
</tr>
<tr>
<td>6.3. Chapter Roadmap</td>
<td>74</td>
</tr>
<tr>
<td>6.4. Doorway Egress</td>
<td>74</td>
</tr>
<tr>
<td>6.5. Doorway Egress Without Additional Capabilities</td>
<td>76</td>
</tr>
<tr>
<td>6.6. Opaque Robots With Lights in ASYNC</td>
<td>77</td>
</tr>
<tr>
<td>6.7. Transparent Robots in ASYNC</td>
<td>98</td>
</tr>
<tr>
<td>6.8. Opaque Robots in SSYNC</td>
<td>113</td>
</tr>
<tr>
<td>6.9. Handling Point Robots on the Half-Plane Line</td>
<td>123</td>
</tr>
<tr>
<td>6.10. Conclusion</td>
<td>125</td>
</tr>
<tr>
<td>Chapter 7. Doorway Egress by Fat Robots</td>
<td>127</td>
</tr>
<tr>
<td>7.1. Contributions of This Chapter</td>
<td>127</td>
</tr>
<tr>
<td>7.2. Chapter Roadmap</td>
<td>127</td>
</tr>
<tr>
<td>7.3. Preliminaries</td>
<td>128</td>
</tr>
<tr>
<td>7.4. Transparent Fat Robots in SSYNC</td>
<td>129</td>
</tr>
<tr>
<td>7.5. Transparent Fat Robots with Lights</td>
<td>141</td>
</tr>
<tr>
<td>7.6. Conclusion</td>
<td>153</td>
</tr>
<tr>
<td>Chapter 8. Doorway Egress Lower Bound Conjecture</td>
<td>154</td>
</tr>
<tr>
<td>8.1. 1-WDE Algorithms</td>
<td>154</td>
</tr>
<tr>
<td>8.2. WDE Algorithms</td>
<td>158</td>
</tr>
<tr>
<td>Chapter 9. Arbitrary Pattern Formation on a Grid</td>
<td>160</td>
</tr>
<tr>
<td>9.1. Contributions of This Chapter</td>
<td>161</td>
</tr>
<tr>
<td>9.2. Previous Work</td>
<td>162</td>
</tr>
<tr>
<td>9.3. Chapter Roadmap</td>
<td>162</td>
</tr>
<tr>
<td>9.4. Preliminaries</td>
<td>163</td>
</tr>
<tr>
<td>9.5. Time Lower Bound</td>
<td>164</td>
</tr>
<tr>
<td>9.6. Classical Model Algorithm</td>
<td>165</td>
</tr>
<tr>
<td>9.7. Distance and Spatial Complexities</td>
<td>177</td>
</tr>
<tr>
<td>9.8. Conclusion</td>
<td>178</td>
</tr>
<tr>
<td>Chapter 10. Concluding Remarks</td>
<td>179</td>
</tr>
<tr>
<td>Bibliography</td>
<td>181</td>
</tr>
<tr>
<td>Vita</td>
<td>189</td>
</tr>
</tbody>
</table>
# List of Tables

5.1 Asynchronous OLCH Algorithm Performance .......................... 40
5.2 Coprime Framework Summary ........................................... 49
5.3 Coprime Framework Algorithm Color Sequences ...................... 51
6.1 Summary of Doorway Egress Algorithms. ............................ 73
7.1 Summary of Doorway Egress Algorithms for Fat Robots. .............. 128
9.1 Arbitrary Pattern Formation Results ................................. 161
List of Figures

2.1 Example Fully-Synchronous Schedule ........................................ 6
2.2 Example Semi-Synchronous Schedule ........................................ 6
2.3 Example Asynchronous Schedule .............................................. 7
2.4 An example of 3 rounds in \( \mathcal{FSYN} \) .................................. 10
2.5 A single epoch in \( \mathcal{ASYN} \) .............................................. 10
4.1 Illustration of CPBCV Phases ................................................. 19
4.2 Safety Triangle (Figure Not to Scale) ...................................... 25
4.3 Corner Insertion ................................................................. 27
4.4 Progression of Pivot Triangles ................................................. 30
4.5 A closer look at two successive pivot triangles. ......................... 31
4.6 Proof of Lemma 4.9. ............................................................. 34
5.1 Effect of the coprimes algorithm framework ............................... 47
5.2 Beacon Robots Moving Into Position ....................................... 49
5.3 Determining \( n \) And \( i \) From the Beacons ............................... 50
5.4 General Procedure for Non-Beacon Robots ................................ 51
5.5 Quadratic Algorithm Example ............................................... 60
5.6 Coprimes Algorithm Example ................................................ 64
6.1 Basic Terminology of Robot Configurations ............................... 76
6.2 Both Cases of Positioning Beacons ......................................... 79
6.3 Movement of Non-Beacons Towards the Beacon ........................... 80
6.4 The Beacon Informs Non-Beacons They Have Reached Unique Lines 81
6.5 Robots Colored \( \text{Lone2} \) Moving to Unique Heights .................. 83
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6</td>
<td>Robot Colored Beacon3 Informs Locking Robot to Move</td>
<td>84</td>
</tr>
<tr>
<td>6.7</td>
<td>Robots Move Horizontally Through the Door</td>
<td>85</td>
</tr>
<tr>
<td>6.8</td>
<td>Locking Robots Moving to Position</td>
<td>99</td>
</tr>
<tr>
<td>6.9</td>
<td>Beacon Robots Moving to Position</td>
<td>100</td>
</tr>
<tr>
<td>6.10</td>
<td>Movement of Non-Beacons Towards the Beacon</td>
<td>101</td>
</tr>
<tr>
<td>6.11</td>
<td>Projection of Non-Beacons Onto Their Initial Line</td>
<td>102</td>
</tr>
<tr>
<td>6.12</td>
<td>Moving Robot From $\ell_1$ Through the Door to Signal Others</td>
<td>104</td>
</tr>
<tr>
<td>6.13</td>
<td>Both Cases of Positioning Beacons</td>
<td>115</td>
</tr>
<tr>
<td>6.14</td>
<td>Movement of Non-Beacons Towards the Beacon</td>
<td>116</td>
</tr>
<tr>
<td>6.15</td>
<td>Robots Move Horizontally Through the Door</td>
<td>117</td>
</tr>
<tr>
<td>6.16</td>
<td>Positioning of the Beacon for Case of Robots on the Half-Plane Line</td>
<td>125</td>
</tr>
<tr>
<td>6.17</td>
<td>Positioning of Non-Beacons for Case of Robots on the Half-Plane Line</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Movement of Robots with Vertical Clearance</td>
<td>132</td>
</tr>
<tr>
<td>7.2</td>
<td>Movement of Robots with Horizontal Clearance</td>
<td>132</td>
</tr>
<tr>
<td>7.3</td>
<td>Movement of Robots in Phase 2</td>
<td>133</td>
</tr>
<tr>
<td>7.4</td>
<td>Movement of Robots in Phase 3</td>
<td>134</td>
</tr>
<tr>
<td>7.5</td>
<td>Movement of Robots with Vertical Clearance</td>
<td>142</td>
</tr>
<tr>
<td>7.6</td>
<td>Movement of Robots with Horizontal Clearance</td>
<td>143</td>
</tr>
<tr>
<td>7.7</td>
<td>Movement of Robots in Phase 2</td>
<td>144</td>
</tr>
<tr>
<td>7.8</td>
<td>Movement of Robots in Phase 3</td>
<td>145</td>
</tr>
<tr>
<td>9.1</td>
<td>Overview of Algorithm 2</td>
<td>167</td>
</tr>
</tbody>
</table>
Symbols and Acronyms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>The center of the doorway</td>
</tr>
<tr>
<td>$d(r_i,c)$</td>
<td>The distance between robot $r_i$ and $c$</td>
</tr>
<tr>
<td>$d_{i,j}(t)$</td>
<td>The distance between $r_i$ and $r_j$ at time $t$</td>
</tr>
<tr>
<td>$\text{dia}(\mathcal{C})$</td>
<td>The diameter (max distance between two robots) of configuration $\mathcal{C}$</td>
</tr>
<tr>
<td>$l_i$</td>
<td>A line passing through $c$ and a robot</td>
</tr>
<tr>
<td>$n$</td>
<td>The number of robots in $\mathcal{C}_0$</td>
</tr>
<tr>
<td>$r_i$</td>
<td>The $i^{th}$ robot</td>
</tr>
<tr>
<td>$s$</td>
<td>Half of the width of the door</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>The algorithm executed by all robots</td>
</tr>
<tr>
<td>$\text{Area}(S)$</td>
<td>The area of the convex hull</td>
</tr>
<tr>
<td>$\mathcal{C}_f$</td>
<td>The final configuration (when all robots have terminated)</td>
</tr>
<tr>
<td>$\mathcal{C}_\text{init}$</td>
<td>The initial configuration</td>
</tr>
<tr>
<td>$\mathcal{C}_\text{pattern}$</td>
<td>The target pattern configuration</td>
</tr>
<tr>
<td>$\mathcal{C}(t)$</td>
<td>The configuration (location and color, if applicable) of robots at time $t$</td>
</tr>
<tr>
<td>$\mathcal{C}_0$</td>
<td>The initial configuration</td>
</tr>
<tr>
<td>$\mathcal{CIR}_i$</td>
<td>The (SEC) containing all $n$ robots of $\mathcal{C}_\text{init}$</td>
</tr>
<tr>
<td>$\mathcal{CIR}_p$</td>
<td>The (SEC) containing all $n$ robots of $\mathcal{C}_\text{pattern}$</td>
</tr>
<tr>
<td>$\mathcal{D}_{\text{avg}}(\mathcal{A})$</td>
<td>The average distance complexity</td>
</tr>
<tr>
<td>$\mathcal{D}_{\text{max}}(\mathcal{A})$</td>
<td>The maximum distance complexity</td>
</tr>
<tr>
<td>$H_a, H_b$</td>
<td>One of the two half-planes</td>
</tr>
<tr>
<td>$H_i$</td>
<td>The height of the $i^{th}$ pivot triangle</td>
</tr>
<tr>
<td>$L_{\text{max}}$</td>
<td>The length of the longest path that a robot takes through all pivot triangles</td>
</tr>
<tr>
<td>$L_{\text{OR}}$</td>
<td>An oriented line of robots</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>A convex polygon</td>
</tr>
<tr>
<td>$\text{Peri}(S)$</td>
<td>The perimeter of the convex hull</td>
</tr>
<tr>
<td>$Q$</td>
<td>The set of $n$ robots</td>
</tr>
<tr>
<td>$\mathcal{S}$</td>
<td>The set of slopes of coprime pairs</td>
</tr>
<tr>
<td>$S(\mathcal{A})$</td>
<td>The spatial complexity of $\mathcal{A}$</td>
</tr>
<tr>
<td>$S(i)$</td>
<td>The slope $g(i)/f(i)$</td>
</tr>
<tr>
<td>$T$</td>
<td>A tree of coprime pairs</td>
</tr>
<tr>
<td>$W$</td>
<td>A wall bisecting the real plane</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASYNC</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>FSYNC</td>
<td>Fully-synchronous</td>
</tr>
<tr>
<td>REC</td>
<td>Smallest grid-aligned rectangle</td>
</tr>
<tr>
<td>SER</td>
<td>Smallest enclosing rectangle</td>
</tr>
<tr>
<td>SEC</td>
<td>Smallest enclosing circle</td>
</tr>
<tr>
<td>SSYNC</td>
<td>Semi-synchronous</td>
</tr>
</tbody>
</table>
Abstract

In recent years, the study of autonomous entities such as unmanned vehicles has begun to revolutionize both military and civilian devices. One important research focus of autonomous entities has been coordination problems for autonomous robot swarms. Traditionally, robot models are used for algorithms that account for the minimum specifications needed to operate the swarm. However, these theoretical models also gloss over important practical details. Some of these details, such as time, have been considered before (as epochs of execution). In this dissertation, we examine these details in the context of several problems and introduce new performance measures to capture practical details.

Specifically, we introduce three new metrics: (1) the distance complexity (reflecting power usage and wear-and-tear of robots), (2) the spatial complexity (reflecting the space needed for the algorithm to work), and (3) local computational complexity (reflecting the computational requirements for each robot in the swarm).

We apply these metrics in the study of some well-known and important problems, such as COMPLETE VISIBILITY and ARBITRARY PATTERN FORMATION. We also introduce and study a new problem, DOORWAY EGRESS, that captures the essence of a swarm’s navigation through restricted spaces. First, we examine the distance and spatial complexity used across a class of COMPLETE VISIBILITY algorithms. Second, we provide algorithms for COMPLETE VISIBILITY on an integer plane, including some that are asymptotically optimal in terms of time, distance complexity, and spatial complexity. Third, we introduce the problem of DOORWAY EGRESS and provide algorithms for a variety of robot swarm models with various optimalities. Finally, we provide an optimal algorithm for ARBITRARY PATTERN FORMATION on the grid.
Chapter 1. Introduction

Distributed computing by autonomous mobile robot swarms is an area that is rapidly developing. Robot swarms offer promising solutions to many current bottlenecks in common sequential processes. Some innovations include search-and-rescue applications, developing smart cities, unmanned vehicles, and various medical applications of nanorobotics. However, many recent advances consider robot swarms with complex, expensive components. While a more powerful swarm (equipped with better sensors, memory, et cetera) may be compelling, distributed problem solving by simple, cheap robots allows for greater scalability and robustness. In this work, we consider the minimal capabilities necessary for robot swarms to accomplish a task. We abstract away the details of physical components and focus on designing the algorithms for these simple robots. That is, we consider only the distributed computing aspect [1].

The various theoretical models for robot swarms capture many practical scenarios, such as fault tolerance and synchronization among the robots. Across the many robot swarm models, the literature has historically focused on providing an algorithm, or algorithms, and proving termination in finite time. That is, much of the work has shown that the swarm will correctly solve the problem eventually. In recent years, however, an emphasis has been placed on providing stronger guarantees on the performance of algorithms executed by the robots. For example, the algorithms may have guaranteed limits on time, number of moves, or computation. The movement towards more practical considerations for robot swarms aims to steer these theoretical algorithms towards more viable strategies in practice.

This dissertation examines various coordination problems, both new and old. There
are many commonly known problems in the literature such as **COMPLETE VISIBILITY** (where all robots position themselves such that all robots can see each other), **GATHERING** (where all robots move to a single location), and **ARBITRARY PATTERN FORMATION** (where all robots position themselves according to any pattern given as input). We also introduce the new problem of **DOORWAY EGRESS** (where robots move to and exit through a doorway). We will formally define problems in Chapter 3 or when needed.

### 1.1. Contributions

In this section, we provide a brief description of the primary contributions of this dissertation, separated by chapter. The details are deferred to the respective chapters.

In Chapter 4, we introduce two new performance metrics for robot swarm algorithms: the distance and spatial complexities. We then apply these metrics to a set of **COMPLETE VISIBILITY** algorithms.

In Chapter 5, we consider **CONVEX HULL FORMATION** algorithms, which are one method by which complete visibility can be achieved. First, we establish lower bounds on time, perimeter, and area. We then present a generic algorithm framework for **CONVEX HULL FORMATION** algorithms. Using this framework, we provide four algorithms, two of which are optimal in time, perimeter, and area. The remaining two algorithms are not optimal in these metrics but use either fewer colors or less local computation during each LCM cycle.

Next, Chapter 6 introduces the problem of **DOORWAY EGRESS** and provides four algorithms for it across a variety of model variations for point (dimensionless) robots. One algorithm is optimal in terms of distance and spatial complexities but not time for any
of the model variations. The remaining algorithms consider a single model variation and provide algorithms that are optimal in time but not distance or space.

Chapter 7 similarly addresses Doorway Egress for fat (unit diameter) robots. This chapter provides two new algorithms that are optimal in time but not distance or space. The linear-time algorithm of the prior chapter, with optimal distance and spatial complexities, immediately extends to fat robots under certain initial configurations as well.

Chapter 8 then provides an overview of our lower bound conjecture for Doorway Egress; We conjecture that, for point or fat robots without any beneficial assumptions, linear time is required. In the chapter, we detail or progress towards a proof for the conjecture.

Finally, Chapter 9 first provides a lower bound on time for Arbitrary Pattern Formation algorithms on the grid. Then, an algorithm for robots with global visibility on the grid that is optimal in terms of time, distance, and space is provided.

1.2. Dissertation Roadmap

First, Chapter 2 provides preliminary information on robot swarm models and the performance metrics we use to analyze them. Chapter 3 examines various coordination problems for robot swarms that have been considered in the literature, and their corresponding results, across a wide array of model variations. Chapters 4 - 9 contain the primary contributions of this work as described in the previous section. Finally, Chapter 10 then re-summarizes the results of this dissertation in further detail and offers a discussion on implications and directions for future work.
Chapter 2. Robot Swarm Models

In this chapter, we will discuss the details of some common robot swarm models. The readers may refer to the following books for further details on various models [1, 2]. We begin by providing an overview of all of the models considered in the dissertation and subsequently explain the areas where they differ. We conclude the chapter with a summary of robot swarm model variations and one example.

2.1. Base Robot Model

We consider a system of \( n \) robots (agents) \( Q = \{r_0, r_1, \cdots, r_{n-1}\} \). Each robot can move in an infinite 2-dimensional real plane \( \mathbb{R}^2 \) or in an infinite 2-dimensional integer plane (or grid) \( \mathbb{Z}^2 \). In the grid setting, each grid point \((i, j)\) has a connection to \((i \pm 1, j)\) and \((i, j \pm 1)\). That is, each point is connected to four other points (e.g. up, down, left, and right). There are no connections \((i, j)\) to \((i \pm 1, j \pm 1)\), that is, no diagonal connections.

In the grid setting, there is no infinite precision even for point robots. The unit distance in the grid reflects both the resolution of distance and speed (distance per unit time) of robots. In either setting, the robots have no access to a common coordinate system and have no inherent knowledge of the value of \( n \).

The robots operate in Look-Compute-Move (LCM) cycles. That is, when a robot is activated, it performs the following steps in order:

- **Look** - At a single instant in time, the robot records the relative positions of all robots visible to it. We call the information received in this step the robot’s *snapshot*.

- **Compute** - The robot uses its snapshot as input to the algorithm that it is executing. This determines an action for it to take, such as a position to move to.

- **Move** - The robot performs the action determined in the compute step.
Immediately after completing one LCM cycle, a robot will either become inactive for a period of time, or they will immediately begin the next LCM cycle. The start times and durations of LCM cycles across all robots are determined by the activation schedule which we detail in Section 2.2.

Robots in the base model are:

**Autonomous:** They operate without any external control.

**Anonymous:** They have no unique identifiers, e.g., names.

**Indistinguishable:** They possess no external distinctions.

**Disoriented:** They do not agree on the direction or scale of location coordinate systems.

**Oblivious:** They retain no memory from prior LCM cycles.

**Silent:** They have no ability to communicate with other robots.

### 2.2. Activation Schedules

An activation schedule for an execution of an algorithm is the set of start times and durations for each look, compute, and move for each robot. In the literature, there are three commonly used activation schedules: FSYNC [3, 4], SSYNC [5, 6], and ASYNC [7, 8, 9].

In the fully synchronous setting (FSYNC), every robot is activated every LCM cycle. Further, all robots begin and end each phase of an LCM cycle (Look, compute, and move) at exactly the same times. Immediately after the conclusion of the “Move” phase of an LCM cycle, the “Look” phase of the next LCM cycle follows. That is, no robots have any period of inactivity. Figure 2.1 provides an illustrative example for a set three robots, \{r_1, r_2, and r_3\}, operating in FSYNC. In the example, the x-axis is indicative of time.
Recall that, although not shown in the figure for the sake of simplicity, the “Look” phase occurs at a single instant in time.

**Recall that, although not shown in the figure for the sake of simplicity, the “Look” phase occurs at a single instant in time.**

<table>
<thead>
<tr>
<th>R1</th>
<th>Look</th>
<th>Compute</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>Look</td>
<td>Compute</td>
<td>Move</td>
</tr>
<tr>
<td>R3</td>
<td>Look</td>
<td>Compute</td>
<td>Move</td>
</tr>
</tbody>
</table>

Figure 2.1. Example Fully-Synchronous Schedule

In the semi-synchronous setting (SSYNC), at least one robot is active in every LCM cycle. Over an infinite number of semi-synchronous LCM cycles, each robot is activated infinitely many times. While a subset of robots may be inactive, the timing of LCM cycles is as in the fully-synchronous case, where each phase of the LCM cycle begins and ends at the same time across all robots.

<table>
<thead>
<tr>
<th>R1</th>
<th>Look</th>
<th>Compute</th>
<th>Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>Look</td>
<td>Compute</td>
<td>Move</td>
</tr>
<tr>
<td>R3</td>
<td>Look</td>
<td>Compute</td>
<td>Move</td>
</tr>
</tbody>
</table>

Figure 2.2. Example Semi-Synchronous Schedule

In the asynchronous setting (ASYNC), robots have no common notion of time. There is no consistency between start times or durations of phases of LCM cycles across robots. As in the semi-synchronous case, robots may be inactive for periods, but each robot is activated infinitely many times over an infinite period.
Note that \( FSYNC \) is a special case of \( SSYNC \) where every robot happens to be active every LCM cycle. Similarly, both \( FSYNC \) and \( SSYNC \) are special cases of \( ASYNC \) such that every robot just so happens to begin and end each phase of their LCM cycles at the same time (with all robots always active).

### 2.3. Memory

There are three primary types of memory in the robot swarm models. The first is obliviousness [3, 10, 11], as described in the base model. Recall that oblivious robots retain no memory (e.g., snapshots, state) from prior LCM cycles.

In some cases, we relax the assumption of obliviousness and silence by assuming that each robot is equipped with a light that can assume a distinct color at a time from a fixed, constant-sized color set. This light can be observed by the robot displaying the light and by the other robots that can see it during a robot’s “Look” phase of an LCM cycle. That is, a robot’s snapshot captures both positions and colors of the robots visible to it. The colors persist from one LCM cycle to another unless changed during a robot’s “Move” phase of an LCM cycle. Except for the lights, the robots are oblivious as in the classical model. This forms the so-called “robots with lights model” [12, 13, 14].

Finally, the robots may have internal bits of persistent memory that function exactly as the lights except that they cannot be observed by other robots [15]. However, we do not consider this case in this dissertation.
2.4. Visibility

In this dissertation, we assume that robots are capable of arbitrary viewing distances. Note that some of the literature has considered robots with limited viewing ranges. That is, a robot’s snapshot will not contain information of any robot that is more than a set distance from its position.

In this work, the robots can either be opaque [13, 16, 17] or transparent [6, 18]. First, we consider transparent robots. In this case, each robot sees all others at all times. Therefore, robots can always determine $n$, the size of the swarm. Since a robot’s snapshot always contains the position of all robots, we say that transparent robots have global visibility.

In the case of opaque robots, if three or more robots are collinear, at least one robot will have its view of another robot obstructed. Therefore, we say that opaque robots have obstructed visibility. Further details on visibility for opaque robots differs based on the extent of the robots, which is explained in Section 2.5.

2.5. Extent

We consider two cases for the extent, or size, of the robots. The most common assumption is that the robots are punctiform, i.e., they are modeled as dimensionless points [6, 19]. If point robots are opaque, a robot $r_i$ can see another robot $r_j$, and vice-versa, iff (if and only if) there is no third robot $r_k$ on the line segment joining $r_i$ and $r_j$. We will use $r_i$ to denote both the robot and its position. Two point robots are said to collide if, under any scheduler, their paths of movement cross or are coincident. All algorithms we discuss in this dissertation are guaranteed to be collision-free. Robots are initially placed
arbitrarily but in unique positions.

We also consider the case of so-called “fat” robots [20, 21, 22]. That is, the robots are discs of diameter 1. Two opaque, fat robots, \( r_i \) and \( r_j \), can see each other iff there exists a point \( p_i \) on the boundary of \( r_i \) and a point \( p_j \) on the boundary of \( r_j \) such that there is no third robot \( r_k \) on the line segment joining \( p_i \) and \( p_j \). This implies that the robots are equipped with 360° of vision. We will use \( r_i \) to denote both the robot and the position of its center point. Two fat robots are said to collide if, under any scheduler, their paths of movement overlap in a 2-dimensional region. This allows robots to move in parallel along paths that meet at a single boundary but do not overlap further. As in the point-robots model, robots are initially placed arbitrarily in non-overlapping (even partially) positions.

Robots’ movements, both for fat and point robots, are assumed to be monotonic. That is, the robots will strictly decrease their distance to their target point of a movement as they perform any movement. Fat robots are said to move along, or to, lanes, which are the Minkowski sum of a line and the unit disk. That is, a lane is an infinite, straight path of unit thickness. The portion of a lane in which a fat robot moves in a single LCM cycle is its path.

2.6. Performance Measures

The goal of this dissertation is to provide algorithms with favorable performance measures using the weakest model possible. A weaker model is one with fewer assumptions or “capabilities.” For example, oblivious robots are less capable than robots with lights and are said to have fewer assumptions. The primary performance measures we use are time, distance, and space. We begin by describing time.
For the \texttt{FSYNC} setting, time is measured in \textit{rounds} of LCM cycles. Figure 2.4 provides an illustration of 3 rounds separated by vertical red lines.

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\texttt{R1} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\texttt{R2} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\texttt{R3} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\end{tabular}
\caption{An example of 3 rounds in \texttt{FSYNC}}
\end{figure}

However, in the \texttt{SSYNC} and \texttt{ASYNC} settings, a robot could remain inactive for an indeterminate amount of time. Therefore, we use the notion of \textit{epochs} to measure time, where an epoch is the smallest time interval during which each robot is active at least once. Therefore, in \texttt{FSYNC} a round is equivalent to an epoch. We will use the term “time” generically to mean rounds for \texttt{FSYNC} and epochs for \texttt{SSYNC} and \texttt{ASYNC}.

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\texttt{R1} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\texttt{R2} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\texttt{R3} & \text{Look} & \text{Compute} & \text{Move} \\
\hline
\end{tabular}
\caption{A single epoch in \texttt{ASYNC}}
\end{figure}

In Figure 2.5, the vertical purple line denotes the end of the first epoch, as it is the first moment at which every robot has finished at least one LCM cycle.

We also consider the distance traveled by the robots when executing an algorithm. Specifically, we consider upper and lower bounds on the maximum distance and the average distance traveled by robots when executing an algorithm. Similarly, we consider upper and lower bounds on the maximum space occupied, relative to the space initially occupied.
by the swarm, when executing an algorithm. These distance and spatial complexities are defined formally in Chapter 4.

2.7. Model Selection

We have described many possible choices when selecting a robot swarm model. A non-exhaustive summary of common model variants is as follows:

- Vision type can be one of: \{Global (Transparency), Obstructed (Opaqueness)\}
- Vision range can be one of: \{Limited, Infinite\}
- Persistence of memory can be one of: \{None, Internal, Lights\}
- The extent, or size, or robots can be one of: \{Point, Fat\}
- The activation schedule can be one of: \{FSYNC, SSYNC, ASYNC\}
- The setting can be one of: \{Continuous (\(\mathbb{R}^2\)), Discrete (\(\mathbb{Z}^2\))\}

From just these options, there are 144 different model variants that can be created. Furthermore, many additional options appear in the literature that do not appear in this dissertation.

Perhaps the most common model considered in the literature is the “classical model”. The classical model uses transparent, point robots with infinite vision ranges and no persistent memory. The classical model is also typically associated with the continuous setting while the activation schedule varies depending on the algorithm.
Chapter 3. Literature Review

In this chapter, we review previous work relating to problems addressed in this dissertation and the area in general. We primarily aim to give a sense of the broad issues addressed in the robot swarm literature, and we add definitions for problems that are examined in this dissertation. The literature survey of the models themselves is in Chapter 2.

Flocchini et al. and Prencipe [23, 24] provide a comprehensive discussion on oblivious mobile robots, including different models, problems, and techniques. A large variety of problems have been addressed for autonomous robot swarm models, including Gathering, Convergence, Arbitrary Pattern Formation, Dispersal, and several other coordination problems [25, 26, 27, 28, 29, 30, 31, 32].

We begin by discussing some fundamental problems for robot swarms. The problem of Gathering, where robots converge to a single location [33, 34, 35], is foundational for robot swarms. One interesting result from [33] showed that Gathering is possible in \( \mathcal{ASYNC} \) for \( n \geq 2 \) transparent robots by only using the assumption that robots agree on the direction and orientation of one axis. This holds even in the presence of faulty robots. This is of note due to the fact that Gathering is deterministically unsolvable without some agreement on the coordinate systems, even in \( \mathcal{SSYNC} \) [2]. Although the additional assumptions are minimal, only guarantees of termination in finite time have been provided. Some prior work has attempted to minimize the distance moved by robots Gathering on grids and to characterize all gatherable configurations [36]. More recent work on the grid has provided an algorithm for Gathering in \( O(n^2) \) time in \( \mathcal{FSYNC} \) [37].

**Definition 1** Let the problem of positioning robots such that every pair of robots can see each other be called Complete Visibility. □
Definition 2  Let the problem of positioning robots such that each robot is located at a distinct corner point of a convex hull (with no robot in the interior or a side of the convex hull) be called Convex Hull Formation.

As stated in Definition 1, the problem of Complete Visibility is for all robots to position themselves such that they are all mutually visible. In the context of opaque, point robots, this means that no three can be collinear. One of the most common methods of achieving this is to position all robots on corners of a convex polygon, or convex hull. This is known as Convex Hull Formation (see Definition 2). Di Luna et al. [38] gave the first Convex Hull Formation algorithm for robots with lights. Subsequent results [39, 40] minimized the number of colors. Recently, attention regarding Complete Visibility has turned to runtime analysis. Vaidyanathan et al. [41] presented an algorithm with runtime $O(\log n)$ using 12 colors in the fully synchronous setting. Sharma et al. [42] presented an algorithm with runtime $O(1)$ using 12 colors in the semi-synchronous setting. Sharma et al. [43, 44] then presented an algorithm with runtime $O(1)$ using 47 colors in the asynchronous setting.

Definition 3  Let the problem of positioning robots such that they form an arbitrary pattern that is provided as input be called Arbitrary Pattern Formation.

Another commonly studied problem is Arbitrary Pattern Formation (APF) [45, 46, 6], which is defined in Definition 3. APF has been the subject of intensive research in the Euclidean plane. Vaidyanathan et al. [6] presented two APF algorithms running in $SSYNC$ for point opaque robots with lights and transparent robots with no lights. Both algorithms run in $O(T_{LE})$, where $T_{LE}$ is the time needed to solve Leader Election, which is the problem of selecting a single robot to be unique from the rest. They further
show that APF can be solved by robots with lights in $\mathcal{ASYNC}$ in $O(T_{LE} + \log(n))$ time. Bose et al. [45] subsequently considered APF on the grid. They show that APF by transparent robots on the grid is deterministically solvable in $\mathcal{ASYNC}$ from any asymmetric initial configuration. However, they provide no analysis on time or distance. Cicerone [47] et al. considered APF in regular tessellation graphs beyond square grids, such as triangular and hexagonal grids, and provided an algorithm that works in any regular tessellation graph. However, they only provided a proof of correctness, and there was no analysis of runtime or number of moves. Additionally, their algorithm only works for asymmetric initial configurations like [45].

Another common problem is that of FILLING (or UNIFORM DISPERAL) [48, 49, 50, 51, 52, 53]. In this problem, robots enter an unknown area (orthogonal grid or arbitrary graph) through one or multiple doors and then disperse across that area to occupy the points (graph nodes). A door is a node in the undirected network from which robots enter the area. The robots are all initially placed at the door and must then move through the target area during dispersion while avoiding collisions. FILLING has been considered on both classical as well as robots with lights model with limited visibility capabilities (in terms of the number of hops) but with or without communication range capabilities (also in terms of number of the hops) [50].

A similar class of problems is UNIFORM SCATTERING [54, 55, 56], where the robots uniformly spread out across an area that they are initially placed in, and therefore do not begin by entering the area through a door. The orthogonal grid setting is typically considered as the problem easily becomes unsolvable in arbitrary graphs.

Others have considered continuous time algorithms, where the robots perpetually
and simultaneously look, compute, and move. This was first considered in [57], where the robots gather using a continuous-time solution. One problem that is commonly associated with continuous algorithms is Flocking [58]. Inspired by biological phenomena, Flocking is the problem of designing self-organizing robots to form a pattern and maintain it while in motion. A similar problem is Guided Flocking, where the flock consists of a single leader and the remaining robots that follow it [1].

The majority of robot swarm algorithms only guarantee that the algorithm will terminate in finite time. There is relatively little work that considers the asymptotic complexity of time for these algorithms [59, 60]. Vaidyanathan et al. [41] presented a Complete Visibility algorithm in $\mathcal{FSYNC}$ that runs in $O(\log n)$ time. Sharma et al. [42] presented an algorithm for Complete Visibility with $O(1)$ runtime in $\mathcal{SSYNC}$. Sharma et al. [43, 44] then presented a $O(1)$ time algorithm in $\mathcal{ASYNC}$. 
Chapter 4. Distance and Spatial Complexity

Traditionally, the goal when designing robot swarm algorithms has been to provide guarantees of termination in finite time using the weakest model possible. However, some recent works have provided analyses on time [16, 9, 6], and a few have provided analyses on the maximum number of moves performed by any robot [61]. In this chapter, we introduce two new performance measures, the distance and spatial complexities.

The distance complexity is an important consideration as physical robots in a swarm are considerably affected by the distance they travel (impacting the time, energy used, and component wear-and-tear). Since robots operate on the real plane (with infinite precision in terms of how close together or spread apart they could be), it is more meaningful to talk of the distance traveled in terms of the initial distance between robots (or the “diameter” of the initial configuration). The idea of distance covered has been considered before, particularly in the context of the gathering problem [62, 63, 64]. Bhagat et al. [13] considered the number of moves by a robot but not the distance covered. Distance is considered in a different way when the robots operate on a graph (as opposed to the real plane) [65, 66].

We also consider the area needed (relative to the initial area occupied by the robots) for the algorithms to run. This “spatial complexity” is an important measure of the algorithm’s ability to allow the swarm to operate within tight spaces. The ability to operate within spatial constraints may be important for a variety of applications such as indoor navigation and search-and-rescue operations.

We examine the distance and spatial complexities in the context of the problem of Complete Visibility [38, 16] in which robots, starting at arbitrary initial positions,
must place themselves in a position in which each pair of robots is within the line of sight of each other. That is, assuming opaque robots, the final configuration has no three robots that are collinear. This problem is fundamental as from a position of complete visibility, several other problems can be more easily solved (such as leader election and pattern formation \[6\]). Specifically, we will consider a class of \textsc{Complete Visibility} algorithms on the robots with lights model in which complete visibility is achieved by placing all \(n\) robots on the corners of an \(n\)-sided convex polygon \([19, 9]\); we will refer to these algorithms as “convex polygon based complete visibility algorithms.”

4.1. Contributions of This Chapter

In this chapter, we formally define the “distance complexity” of an autonomous robots algorithm. We then study the distance complexity of convex polygon based complete visibility algorithms. We ultimately show that the convex polygon based complete visibility algorithms we examine all work with an optimal “distance complexity” of \(O(1)\).

We similarly define the “spatial complexity”, which reflects the area needed (relative to the initial area occupied by the robots) for the algorithms to run. We prove that the spatial complexity of the class of \textsc{Complete Visibility} algorithms that we examine is also optimal at \(O(1)\).

We then extend these results to leader election and gathering, showing that they can be solved in the robots with lights model with \(O(1)\) distance and spatial complexities.

4.2. Chapter Roadmap

We discuss the model and preliminaries, primarily regarding the problem of complete visibility, in Section 4.3. Section 4.4 introduces the concept of distance complexity
and Section 4.5 provides a distance complexity analysis for a set of Complete Visibility algorithms. Section 4.6 similarly introduces the concept of spatial complexity provides a spatial complexity analysis for the algorithms discussed. Section 4.7 describes how the results for Complete Visibility extend immediately to other problems. Section 4.8 then provides a brief discussion.

4.3. Convex Polygon Based Complete Visibility

A polygon is convex if and only if the line segment joining any two points in or on the polygon lies entirely in or on the polygon. A polygon consists of edges, or sides, that meet at vertices which we call corners. Figure 4.1(c) shows an example of a convex polygon. If robots are positioned at the vertices of a convex polygon, then they are completely visible to each other. However, we recognize that Complete Visibility does not require this approach. In this chapter, we consider this class of “convex polygon based complete visibility algorithms.” Each algorithm in this class typically has the following three phases:

- **Phase 0** (Delinearization): A set of points that are not collinear has a well-defined convex hull. The goal of this phase is to perturb robots out of a collinear initial configuration so that they can start with a convex hull. If all \( n \) robots are in a collinear initial configuration, then a subset of robots move a small constant distance (specifically, they move perpendicular to the line on which they lie), thus forming a non-collinear configuration with a well-defined convex hull. Figure 4.1(a,b) illustrates this.

- **Phase 1** (Interior Depletion): By the start of this phase, all robots are on the perimeter or inside of a convex polygon. The goal of this phase is to move all of the robots from the interior of the convex polygon to the perimeter. This results in a configuration such that all robots are either corners of the convex polygon or lie along its edges. (see Figure 4.1(c)).

- **Phase 2** (Edge Depletion): Robots on the sides of the convex polygon will move outward (although one algorithm moves them inwards instead) to become corners
of a new convex polygon. This may iterate multiple times until all robots lie on vertices of a convex polygon (see Figure 4.1(d)). This configuration is sufficient for complete visibility.

Particular algorithms considered in this chapter themselves differ in the techniques used and in the time taken for these phases. In particular, the algorithm of Sharma et al. [5, 9] Vaidyanathan et al. [16, 67] and Di Luna et al. [38, 19] run in constant, $O(\log n)$ and $O(n)$ time, respectively.

4.4. Distance Complexity

Let $Q = \{r_i : 0 \leq i < n \}$ be the set of robots. In this chapter, the position of robot $r_i$ is the only meaningful state of the robot that we will be concerned with. Although the robot itself is only aware of its position relative to other robots, we consider
global coordinates for the analysis. The configuration is given by \( \mathcal{C} = \{ p_i : 0 \leq i < n \} \), where \( p_i = (x_i, y_i) \). Let \( d_{i,j}(\mathcal{C}) \) be the distance between robots \( r_i \) and \( r_j \) in configuration \( \mathcal{C} \). When there is no danger of confusion, we will drop the (\( \mathcal{C} \)) attribute.

**Definition 4** Let the diameter of a configuration \( \mathcal{C} \) be the largest of the distances between the robots in that configuration. That is, \( \text{dia}(\mathcal{C}) = \max \{ d_{i,j} : 0 \leq i, j < n \} \). \( \square \)

For a given algorithm \( \mathcal{A} \), a given initial configuration \( \mathcal{C}_0 \), and a final configuration \( \mathcal{C}_f \) that \( \mathcal{A} \) takes the robots to, let robot \( r_i \) move a total distance of \( \delta_i \) as it goes from \( \mathcal{C}_0 \) to \( \mathcal{C}_f \). Clearly \( \delta_i \) is a function of \( \mathcal{A}, \mathcal{C}_0, \mathcal{C}_f \) (although the notation for \( \delta_i \) does not show this for clarity).

Let the distance moved between the initial and final configurations, averaged over all the robots, be

\[
\delta_{\text{avg}}(\mathcal{A}, \mathcal{C}_0, \mathcal{C}_f) = \text{Ave}_i(d_i).
\]

Let the maximum distance moved by any robot between the initial and final configurations be

\[
\delta_{\text{max}}(\mathcal{A}, \mathcal{C}_0, \mathcal{C}_f) = \text{Max}_i(d_i).
\]

We now define two distance complexity measures for the algorithm.

The average distance complexity of algorithm \( \mathcal{A} \) is

\[
D_{\text{avg}}(\mathcal{A}) = \max_{\mathcal{C}_0, \mathcal{C}_f} \left[ \frac{\delta_{\text{avg}}(\mathcal{A}, \mathcal{C}_0, \mathcal{C}_f)}{\text{dia}(\mathcal{C}_0)} \right].
\]

The average distance complexity of \( \mathcal{A} \) is the distance moved by the robots on average, expressed in terms of the diameter of the initial configuration. This “initial diameter” is necessary as point robots on a real plain allow any configuration to be expanded or contracted arbitrarily without affecting the execution of the algorithm. Further, the distance
complexity considers the worst possible initial and final configurations. Without this, an initial configuration that is the same as the final configuration will give a distance complexity of 0.

In a similar manner, the maximum distance complexity of algorithm $\mathcal{A}$ is

$$D_{\text{max}}(\mathcal{A}) = \max_{C_0, C_f} \left[ \frac{\delta_{\text{max}}(\mathcal{A}, C_0, C_f)}{\text{dia}(C_0)} \right].$$

It should be noted that the distance complexity can take many forms. In general, with an initial diameter of $D$, an algorithm could (a) cause no robot movement resulting in a distance complexity of “0,” or (b) cause the robots to move a constant distance each (independent of both $n$ and $D$) resulting in a “decreasing” distance complexity $\Theta\left(\frac{1}{D}\right)$, or (c) cause robots to move distances that are functions of $n$ and/or $D$; here the distance complexity could be a function of $n$ and $D$.

We now place these ideas in the context of convex polygon based complete visibility algorithms.

**Conjecture 4.1** Every convex polygon based complete visibility algorithm has a distance complexity of $\Omega(1)$.

**Proof approach:** Arrange the $n$ robots in an initial configuration so that they are placed in two concentric circles of diameter $D$ and $\frac{D}{2}$ with $\frac{n}{2}$ robots uniformly placed on each circle. We conjecture that to reduce the average distances moved, the final $n$-corner convex polygon must occupy the area between the two circles. If so, it can now be shown that the sum of the minimum distances from the robots’ initial positions to the final convex hull is $\Omega(nD)$, establishing an average (and hence maximum) distance complexity of $\Omega(1)$. □
We will show that the convex polygon based complete visibility algorithms that we study in this chapter all have a maximum (and hence, average) distance complexity of $O(1)$, which if Conjecture 4.1 is true, is optimal.

4.5. Distance Analysis

In this section, we analyze convex polygon based complete visibility algorithms to determine their distance complexity. Along the way, we also establish the optimality of the distance complexity of the algorithms studied.

We now go through the three phases (see Section 4.3) of these algorithms and show that each one of them has a maximum distance complexity of $O(1)$. Before we proceed, we observe that for all of the convex polygon based complete visibility algorithms [38, 16, 19, 9, 5, 67] that we study, no robot moves a distance of greater than $D$ (the initial configuration diameter) at any step. This implies that the constant time algorithm of Sharma et al. [9, 5] has a constant distance complexity. In the remaining discussion, we will deal more specifically with the algorithms of Vaidyanathan, Sharma et al. [16, 67] and Di Luna et al. [38, 19] that run in non-constant time; however, our discussion also applies to the techniques used in the constant time algorithm of Sharma et al.

4.5.1. Phase 0: Delinearization

Recall that the goal of Phase 0 is to take a linear arrangement of robots and make it a nonlinear configuration that has a convex hull. A subset of robots (for example, the endpoint robots) move a small distance perpendicular to the line on which they lie. We will make a change to the algorithms such that instead of moving a small constant distance, each moving robot will travel a distance not exceeding the distance between itself
and its furthest visible neighbor. Their movement breaks the collinearity of the initial configuration. Immediately upon completion of this phase (whether or not robots must actually move) it is clear that a convex polygon has been formed.

Let $D$ be the diameter of the initial configuration before Phase 0. Clearly, the maximum distance that a robot may move is $D$, and the diameter of the resultant configuration at the end of Phase 0 would be $\Theta(D)$.

With no loss of generality, let us call the diameter of the configuration at the end of Phase 0 as simply $D$.

**Lemma 4.2** The maximum (and hence, average) distance complexity of Phase 0 of any convex polygon based complete visibility algorithm is $O(1)$. □

### 4.5.2. Phase 1: Interior Depletion

At the start of Phase 1, all robots are at the corners, on sides, or in the interior of a convex polygon of diameter $D$. The goal of Phase 1 is to move all interior robots to the sides.

In the algorithms we consider, every robot in the interior moves a constant $c \geq 1$ number of times within the convex polygon in order to get to a side. Clearly, these movements cannot cover a distance greater than $cD$, where $D$ is the diameter of the convex polygon.

**Lemma 4.3** The maximum (and hence, average) distance complexity of Phase 1 of any convex polygon based complete visibility algorithm is $O(1)$. □

Unlike usual complexity measures, the distance complexity can be a decreasing function or even 0, where the robots move $o(D)$ distance (or possibly not move at all).
Therefore, a $O(1)$ distance complexity is not necessarily the best possible. However, we show now that there exists a configuration for which the average (and hence, maximum) distance complexity for Phase 1 is $\Omega(1)$.

Let the convex polygon at the start of Phase 1 be an equilateral triangle with three robots at its corners and the remaining $n - 3$ robots clustered in the interior infinitely close to the centroid of the equilateral triangle. The diameter $D$ of this configuration is the side length of the triangle. The shortest distance from the centroid to a side is $\hat{D} = \frac{D}{2\sqrt{3}} = \Theta(D)$. Therefore, each of the $n - 3$ interior robots moves a distance of about $\hat{D}$, or more, and the average distance moved is about $\left(\frac{n-3}{n}\right)\hat{D}$ or more. Thus the average distance complexity is $\Omega\left(\frac{(n-3)\hat{D}}{nD}\right) = \Omega(1)$.

**Lemma 4.4** The average (and hence, the maximum) distance complexity of Phase 1 is $\Omega(1)$. □

4.5.3. Phase 2: Edge Depletion

Recall that at the start of Phase 2, every robot is either at a corner of the convex hull or on one of its sides. The goal of this phase is to move the side robots outward to new corners; the corner robots do not move for the rest of the algorithm. Phase 2 of all of the convex polygon based complete visibility algorithms builds on the idea of a “safety triangle” that allows side robots to move to corners of a new polygon in a manner that (a) keeps corners of the existing polygon as corners of the new polygon, and (b) keeps the new polygon convex; we will talk about safety triangles in more detail below. The different algorithms we consider move side robots at different times and in slightly different ways in Phase 2, but the underlying idea of all of these safety triangle based movements is sim-
ilar. As noted earlier, the constant time algorithms need no additional consideration to establish a $O(1)$ distance complexity. In this section, we will consider the $O(\log n)$-time algorithm of Vaidyanathan et al. [16], that uses a procedure called “corner insertion” to complete Phase 2. While the algorithm of Di Luna et al. [38, 19] runs in $O(n)$ time, and the corner robots move inwards, the ideas in this section apply to it as well. Before we proceed, we discuss a few additional ideas regarding edge depletion.

### 4.5.3.1. Safety and Pivot Triangle

Consider a side $S$ of a polygon $P_0$ (see Figure 4.2). Let $S$ make interior angles $\theta_0$ and $\phi_0$ with its neighboring sides at corners $c_1, c_2$, respectively. Because $P_0$ is convex, $\theta_0, \phi_0 < \pi$. In Phase 2, side points move to corners of a new convex polygon $P_1$; for example, a side point on side $S$ in Figure 4.2 moves to a new point $x$ (that forms a corner of $P_1$). The question is, how “far” from the side $S$ can the point $x$ be?

![Figure 4.2. Safety Triangle (Figure Not to Scale)](image)

Define the safety angle of corner $c_1$ (with interior angle $\theta_0$) to be

$$\theta' = \frac{\pi - \theta_0}{4} \quad (4.1)$$

Notice that because $0 < \theta_0 < \pi$, the safety angle satisfies $0 < \theta' < \frac{\pi}{4}$. Each side $S$ and its
neighboring side on corner $c_1$ has an associated safety angle $\theta'$ (see Figure 4.2). Corner $c_2$ with interior angle $\phi_0$ has a safety angle $\phi' = \frac{\pi-\theta_0}{4}$. The triangle $\langle c_1, c_2, h \rangle$ (see Figure 4.2) is called the safety triangle of side $S$. In a similar manner, the side adjacent to $S$ on corner $c_1$ also has a safety triangle.

To see the significance of the safety triangle, observe that the interior angle at corner $c_1$ that is bounded by sides of the two safety triangles is

$$\theta_0 + 2\theta' = \theta_0 + \frac{\pi - \theta_0}{2} = \frac{\theta_0 + \pi}{2} < \pi$$

Thus, if point $x$ was within the safety triangle, then it would guarantee that the new polygon $P_1$ is convex and also has $c_1$ as one of its corners.

The convex polygon based complete visibility algorithms require that angle $\theta_1$ (used to reach point $x$) be within the safety triangle. The triangle $\langle c_1, x, c_2 \rangle$ will be called the pivot triangle of $S$ (see Figure 4.2) and the angle $\theta_1$ is the pivot angle of corner $c_1$. To satisfy the requirement of convex polygon based complete visibility algorithms that the pivot triangle lie within the safety triangle, we will assume that the pivot angle $\theta_1 = \epsilon \theta'$, for some $0 < \epsilon < 1$. Ultimately we will set $\epsilon = 1$ to reflect a worst-case scenario.

4.5.3.2. Corner Insertion

The core substep of Phase 2 in the algorithm of Vaidyanathan et al. [16] is called “corner insertion.” We outline the functionality of this substep to help with the subsequent distance analysis. The goal of a corner insertion step is to take a side $S$ of polygon $P_0$ with (say) $m \leq n - 3$ side robots and convert it to two sides, each with approximately $\frac{m}{2}$ side robots, of a new convex polygon $P_1$. This is done by first moving a side robot to point $x$ (see Figure 4.2) within the safety triangle. This robot will become a corner robot
for the next polygon $P_1$ after corner insertion. The other side robots of the original polygon $P_0$ now move to become side robots of $P_1$. This is illustrated for one half of the safety triangle of $S$ in Figure 4.3.

![Figure 4.3. Corner Insertion](image)

For example, the side robot of $P_0$ that is at position $x_1$ now moves to position $x_2$ on a side of $P_1$. Other side robots of $P_0$ move similarly. Further, other sides of $P_0$ concurrently perform similar corner insertions in the formation of $P_1$.

In the next iteration, corner insertion is performed on the sides of $P_1$ to form another convex polygon $P_2$. For example, the robot at $x_2$ now moves to $x_3$. This process continues (for at most $\log n$ iterations) until no side points remain. Figure 4.3 shows the robot movement in each iteration of corner insertion.

We now make a few observations about corner insertion that will be useful in the subsequent discussion.

For any iteration of corner insertion, let the robot that becomes a new corner of a given side be called its “pivot” and let the final position of this robot be called the “pivot position” of the side. In Figures 4.2 and 4.3, the pivot position of side $S$ is at $x$.

Although we have not indicated so in the figures, the pivot moves a constant num-
ber of times within the safety triangle before it comes to the pivot position as a corner. However, once it reaches this position and becomes a corner, it does not move in subsequent corner insertions. Therefore we have the following:

**Observation 1** The distance moved by the pivot of side $S$ is of the same order as the distance of the pivot position from side $S$.

Consequently, in the subsequent discussion, we will ignore the additional steps taken by the pivot.

**Observation 2** All movements in corner insertion for a side $S$ are perpendicular to $S$ (to within the exception noted in Observation 1).

Consider corner insertion for some side $S$ that converts polygon $P_i$ to $P_{i+1}$. Let $x_i$ and $x_{i+1}$ denote the position a side robot of $P_i$ and $P_{i+1}$, respectively. Call $x_i$ the projection of $x_{i+1}$ on $S$. In Figure 4.3, $x_1$ is the projection of $x_2$ on $S$, $x_2$ is the projection of $x_3$ on side $S_1$ and so on.

Observe that as a side point of $S$ gets closer to the projection of the pivot position of $S$, the distance it travels during the current corner insertion becomes larger; in Figure 4.3 the lines extending from side $S$ are longer towards the right. However, if the point is too far to the right in the figure, then it ends up far from the projection of the pivot point for the next iteration and does not travel far.

**Observation 3** The most total distance (over all iterations) is traversed from some position that is neither too close nor too far from the projection of the pivot point.

Points $x_1, x_2, \cdots, x_5$ illustrate such a path in Figure 4.3.
4.5.3.3. Phase 2 Distance Analysis

Recall that for safety angle $\theta'$ and pivot angle $\theta_i$, we have assumed $\theta_i = \epsilon \theta'$, for some $0 < \epsilon < 1$. Let the interior angle of some corner $c$ of the initial polygon $P_0$ be $\theta_0$. As observed earlier, the corner insertion can occur on one or both sides incident on $c$. We examine the growth of subsequent pivot angles under these two circumstances.

Suppose corner insertion occurs only on one of the sides of $c$. By Equation (4.1), $\theta_1 = \epsilon \theta' = \frac{\epsilon}{4} (\pi - \theta_0)$. Similarly, $\theta_2 = \frac{\epsilon}{4} (\pi - (\theta_0 + \theta_1)) = (1 - \frac{\epsilon}{4}) \theta_1$. Proceeding in the same way $\theta_3 = \frac{\epsilon}{4} (\pi - (\theta_0 + \theta_1 + \theta_2)) = (1 - \frac{\epsilon}{4}) \theta_2$. In general,

$$\theta_i = \frac{\epsilon}{4} \left( \pi - \sum_{j=0}^{i-1} \theta_j \right) = \left( 1 - \frac{\epsilon}{4} \right)^{i-1} \theta_1. \quad (4.2)$$

We now similarly consider the case where corner insertion applies to both sides of corner $c$. Here, $\theta_1 = \frac{\epsilon}{4} (\pi - \theta_0)$ is still the same as in the earlier case. However, $\theta_2 = \frac{\epsilon}{4} (\pi - (\theta_0 + 2\theta_1)) = (1 - \frac{\epsilon}{2}) \theta_1$. Proceeding in the same way $\theta_3 = \frac{\epsilon}{4} (\pi - (\theta_0 + 2\theta_1 + 2\theta_2)) = (1 - \frac{\epsilon}{2}) \theta_2$. In general,

$$\theta_i = \left( 1 - \frac{\epsilon}{2} \right)^{i-1} \theta_1. \quad (4.3)$$

The following lemma is a direct consequence of Equations (4.2) and (4.3).

**Lemma 4.5** A one-sided application of corner insertion to a side incident on a corner of the initial polygon $P_0$ results in larger pivot angles $\theta_i$. \hfill \square

As noted earlier, the longest distance covered by a robot could span through several iterations of corner insertion. Figure 4.4 illustrates a longest path as it winds its way through $k$ pivot triangles (where $1 \leq k \leq \log n$).

Let $A_i$ be the length of this path within the $i^{th}$ pivot triangle and let $H_i$ be the height of the $i^{th}$ pivot triangle (see Figure 4.4). Recall that the path of the robot within
a pivot triangle is perpendicular to the base of the triangle; consequently, $A_i \leq H_i$. Therefore the length of the longest path is

$$L_{\text{max}} = \sum_{i=1}^{k} A_i \leq \sum_{i=1}^{k} H_i$$  \hspace{1cm} (4.4)

In the remainder of this section we will show that $\sum_{i=1}^{k} H_i$, and hence $L_{\text{max}}$ is $O(D)$, where $D$ is the diameter of the initial configuration.

Figure 4.5 shows levels $i$ and $i + 1$ of Figure 4.4 in more detail.

Note first that since $\theta_i, \phi_i < \frac{\pi}{4}$, we have $L_{i+1} < L_i$. In fact in the context of Figure 4.4, we have

$$\text{For all } 1 < i \leq k, \quad L_i < L_1$$ \hspace{1cm} (4.5)

Observe next that $H_i = L'_i \tan \theta_i = L''_i \tan \phi_i = (L_i - L'_i) \tan \phi_i$. Expressed only in
terms of $L_i, L'_i$ we have

$$L'_i = \frac{L_i \tan \phi_i}{\tan \theta_i + \tan \phi_i} \quad (4.6)$$

$$H_i = L'_i \tan \theta_i = \frac{L_i \tan \theta_i \tan \phi_i}{\tan \theta_i + \tan \phi_i} < \frac{L_1 \tan \theta_i \tan \phi_i}{\tan \theta_i + \tan \phi_i} \quad (4.7)$$

**Lemma 4.6** For any $1 \leq i \leq k$, the height of the $i^{th}$ pivot tree is $H_i < \frac{4L_1 \theta_i \phi_i}{\pi(\theta_i + \phi_i)}$.

**Proof:** From Equation (4.7), we have

$$H_i < \frac{L_1 \tan \theta_i \tan \phi_i}{\tan \theta_i + \tan \phi_i} = \frac{L_1}{\frac{1}{\tan \theta_i} + \frac{1}{\tan \phi_i}}$$

In the range $0 < \theta_i < \frac{\pi}{4}$, we have $\frac{\theta_i}{\tan \theta_i} > \frac{\pi}{4}$ or $\frac{1}{\tan \theta_i} > \frac{\pi}{4\theta_i}$. Similarly, $\frac{1}{\tan \phi_i} > \frac{\pi}{4\phi_i}$. Therefore,

$$H_i < \frac{L_1}{\frac{1}{\tan \theta_i} + \frac{1}{\tan \phi_i}} < \frac{\pi}{\frac{4\theta_i}{\theta_i} + \frac{4\phi_i}{\phi_i}} = \frac{4L_1 \theta_i \phi_i}{\pi(\theta_i + \phi_i)}$$

□

From the expression $H_i < \frac{4L_1 \theta_i \phi_i}{\pi(\theta_i + \phi_i)}$ it is clear that as $\theta_i$ (or $\phi_i$) increases, $\frac{1}{\theta_i}$ (or $\frac{1}{\phi_i}$) decreases and the entire upper bound for $H_i$ increases. Therefore we will bound $H_i$ by

![Figure 4.5. A closer look at two successive pivot triangles.](image)
using the largest possible values for $\theta_i$ and $\phi_i$. From Lemma 4.5 and Equation (4.2) we see that $\theta_i = (1 - \frac{\epsilon}{4})^{i-1} \theta_1$. Similarly $\phi_i = (1 - \frac{\epsilon}{4})^{i-1} \phi_1$.

There is another possible value for $\theta_i, \phi_i$. Observe from Figure 4.5 that $\angle(u, x, w) = \gamma_i = \pi - (\theta_i + \phi_i)$. As a result $\phi_{i+1} = \frac{\pi}{4}(\pi - \gamma_i) = \frac{\pi}{4}(\theta_i + \phi_i)$. Similarly for an analogous orientation of corner insertion, we can have $\theta_{i+1} = \frac{\pi}{4}(\theta_i + \phi_i)$. It can be shown inductively that for the case discussed above, $\theta_i, \phi_i = (\frac{\pi}{4})^{i-1} (\theta_i + \phi_i)$.

Therefore we can assert that for all $1 \leq i \leq k$,

\[ \theta_i, \phi_i < \left( (1 - \frac{\epsilon}{4})^{i-1} + (\frac{\epsilon}{4})^{i-1} \right) (\theta_1 + \phi_1) \]

\[ < \frac{\pi}{2} \left( (1 - \frac{\epsilon}{4})^{i-1} + (\frac{\epsilon}{4})^{i-1} \right) = X_i \text{ (say)} \]  

(4.8)

From Lemma 4.6 and Equation (4.8), we have the following bound for $H_i$.

\[ H_i < \frac{4L_1}{\pi} \left( \frac{1}{X_i} \right) = \frac{2L_1X_i}{\pi} \]

\[ = L_1 \left[ (1 - \frac{\epsilon}{4})^{i-1} + (\frac{\epsilon}{4})^{i-1} \right] \]  

(4.9)

We are now in a position to derive the main result of this section.

**Lemma 4.7** The maximum (and hence, average) distance complexity of Phase 2 is $O(1)$.

**Proof:** Recall that the diameter, $D_2$, of the Phase 2 initial configuration is $O(D)$ where $D$ is the initial configuration of the entire algorithm. All we need to show now is that the maximum length path has length $O(D)$.

From Equations (4.4) and (4.9), the longest path length is:
\[ \mathcal{L}_{\text{max}} < \sum_{i=1}^{k} H_i < L_1 \left[ \sum_{i=0}^{\infty} \left( 1 - \frac{\varepsilon}{4} \right)^i + \left( \frac{\varepsilon}{4} \right)^i \right] \]

\[ = L_1 \left[ \frac{1}{\frac{\varepsilon}{4}} + \frac{1}{1 - \frac{\varepsilon}{4}} \right] = \frac{16L_1}{\varepsilon(4 - \varepsilon)} \]

\[ \leq \frac{16D_2}{\varepsilon(4 - \varepsilon)} = O(D) \]

We have used the fact that a side of the polygon has length \( L_1 \) that is less than its diameter \( D_2 \).

The proof would seem to suggest that having a larger value of epsilon is better than having a smaller value.

Using Lemmas 4.2, 4.3 and 4.7 we now have the main result of this section.

**Theorem 4.8** The maximum (and hence, average) distance complexity of convex polygon based complete visibility algorithms is \( O(1) \).

In light of Lemma 4.4 and Conjecture 4.1, one could say that the distance complexity of Theorem 4.8 is likely optimal.

### 4.6. Spatial Complexity

The distance complexity of a robot algorithm reflects the distance covered by the robots in the course of the algorithm. The spatial complexity reflects the amount of room needed for the robots to execute the algorithm. This is an important determinant of the usefulness of an algorithm that may have to be executed in tight spaces (such as in a search and rescue operation). A measure such as the area of the convex hull of a configuration may be a good reflection of the space used by the robots. However, the space available is likely to be more regular in structure (such as a circle).
For any configuration $C$ let $\text{space}(C)$ denote the area of the smallest circle enclosing all robots in that configuration.

**Lemma 4.9** For any configuration $C$ with diameter $D$, $\text{space}(C) \leq \frac{3\pi D^2}{4}$

**Proof:** Let $u, v$ be a pair of robots in configuration $C$ with distance $D$ between them. Every other robot $r$ must be within a distance of $D$ from both $u$ and $v$. The smallest enclosing circle for this area has a diameter of $\sqrt{3}D$ (or a radius of $\frac{\sqrt{3}D}{2}$). Therefore the area of this circle is $\frac{3\pi D^2}{4}$.

![Figure 4.6. Proof of Lemma 4.9.](image)

Consider an algorithm $A$ that runs for initial configuration $C_0$ to a final configuration $C_f$. The Spatial Complexity of the algorithm is

$$S(A) = \max_{C_0 \leq C \leq C_f} \left[ \frac{\text{space}(C)}{\text{space}(C_0)} \right]$$

In the above definition $C_0 \leq C \leq C_f$ indicates every configuration between the initial and
Theorem 4.10  The spatial complexity of convex polygon based complete visibility algorithms is $O(1)$.

Proof. For $0 \leq x < 3$, let the configuration at the start of Phase $x$ be $C_x$ and let its diameter be $D_x$. Observe that if the maximum distance complexity of a Phase $x$ is $\delta_x$, then diameter $D_{x+1} \leq \sqrt{3}(D_x + 2\delta_x D_x) = \sqrt{3}D_x(1 + 2\delta_x)$; here the maximum distance moved is $\delta_x D_x$ and the new longest distance could be stretched by $2\delta_x D_x$ (with two large movements in opposite directions), and finally the new diameter could be extended by a factor of $\sqrt{3}$ as in the proof of Lemma 4.9. If $\delta_x = O(1)$, then $D_{x+1} = O(D_x)$. For all of the phases of the algorithm (Lemmas 4.2, 4.3 and 4.7), $\delta_x = O(1)$. Thus, the algorithm uses $O(\text{space}(C_0))$ area at all points and the spatial complexity is $O(1)$. □

Since initial configuration is included in the spatial complexity, it is always $\Omega(1)$ for every algorithm, and hence the spatial complexity of Theorem 4.10 is optimal.

4.7. Other Applications

Positions of complete visibility allow for other problems to be easily solved. For example, leader election can be solved without further movement after COMPLETE VISIBILITY is solved. Gathering can be solved (on a model with lights), with a constant number of movements within the convex hull of the configuration [6, 19]. Therefore assuming robots initially start at an arbitrary configuration we have the following by-product results.

Theorem 4.11  The Leader Election and Gathering problems can be solved on the robots with lights models with $O(1)$ distance and spatial complexities. □
4.8. Conclusion

In this chapter, we have introduced the ideas of distance and spatial complexities of a robot swarm algorithm. These complexities assess the algorithms’ ability to preserve battery and component life and to operate in small spaces. We have considered a class of Complete Visibility algorithms that are based on positioning robots on corners of a convex hull and proved that all known algorithms in this class have $O(1)$ distance and spatial complexities.

All of the algorithms of this chapter consider point robots on a real plain, in the process using the “infinite precision” available to squeeze robots into small spaces. The algorithms of Vaidyanathan et al. and Di Luna et al. use this ability primarily when the initial configuration is very dense and when the perimeter of the convex hull does not have sufficient room for all robots. In a practical situation in which the robots operate in a sufficiently large space, then these algorithms can be easily modified to accommodate the robots, and the algorithm would run relatively unchanged. However, unlike the other algorithms, Phase 1 of the Gokarna et al. algorithms use the idea of an eligible area that could run into problems even if the robots operate over a large area, but some of the corners have a dense distribution of robots. In this context, the fact that the algorithms of Vaidyanathan et al. are slower is offset by the fact that they exhibit favorable distance and spatial complexities.

As we have noted in Section 4.7, the main results of this chapter enable other results. Using the distance and spatial complexities as considerations in algorithm design for a wide range of problems would serve to amplify the utility of many algorithms.
Chapter 5. Convex Polygon Based Complete Visibility on a Grid

In this chapter, we once again study the fundamental Convex Hull Formation problem. The model we use in this chapter is the robots with lights model [1, 38, 26, 12] (see Section 2.3) that incorporates a limited form of direct communication — each robot has a visible light that can assume colors from a constant-sized set. A fundamental difference between the ideas of this chapter and the previous chapter is the use of the grid, as opposed to the real plane. The main goal of this chapter is to develop efficient algorithms for Convex Hull Formation on an infinite grid in the asynchronous setting.

Let the robots begin in any initial configuration in which they are located at distinct points on the grid. Recall that the problem is to position robots such that each robot is at a distinct corner point of a convex hull in the plane. Accomplishing this, thereby solving Convex Hull Formation, enables solutions to many other robot coordination problems under obstructed visibility including Complete Visibility (also called mutual visible in some cases), pattern formation, Gathering, and Leader Election [6, 43]. Thus, Convex Hull Formation is a fundamental primitive with many uses in autonomous robot coordination.

Convex Hull Formation has received much attention recently. In the robots with lights model, a series of results [41, 42, 44] led to an $O(1)$-time, 47-color algorithm in the 2-dimensional real plane in the asynchronous setting [43]. In this chapter, we consider Convex Hull Formation on a grid that restricts each move of a robot to one of the four neighboring grid points from its current position (grid point).

The field of theoretical computer science has focused heavily on the runtime and memory requirements of algorithms. In the context of distributed computing by oblivious
mobile robots, evaluation of such algorithms often consider only guarantees of termination in finite time. However, recent work has considered more detailed analysis on time, such as by providing upper bounds on execution time. While this provides greater insight into the efficacy of algorithms, there are additional considerations when discussing robot swarm algorithms. We measure the efficiency of a Convex Hull Formation algorithm through the following performance measures.

- **Time**: How long does it take robots to relocate to corners of a convex hull? Time is measured in epochs, where an epoch is a duration in which each robot completes at least one LCM cycle.

- **Distance**: How far do robots move during execution of the algorithm? Distance is measured in units of length on the grid.

- **Perimeter**: How large should the perimeter of the convex hull be? Perimeter is measured in units of length on the grid.

- **Area**: How much surface area should be occupied by the convex hull? Area is measured in squared unit length on the grid.

- **Local Computation**: How long should it take for a robot to complete the “Compute” phase of an LCM cycle?

- **Colors**: How large of a color palette is needed?

Time is a performance metric that has been considered for Convex Hull Formation in the Euclidean plane, while perimeter is a new metric we study in the context of Convex Hull Formation by robots, considered for the first time in Hector et al. [68]. Perimeter and area become important when robots have to arrange themselves in a convex hull configuration satisfying spatial constraints. These measures are similar to the “spatial complexity” introduced by in Chapter 4. Recall that the distance measure is used to describe the energy consumption of robots (see Chapter 4). In one of our algorithms, no robot moves more than unit distance per epoch. In the other algorithms presented in
In this chapter, we establish two results, Results 1 and 2 (see Table 5.1). The results are discussed in detail below. Result 1 establishes lower bounds, and Result 2 establishes three algorithms which decompose Convex Hull Formation into two subproblems: Oriented Line Formation and Oriented Line to Convex Hull. The Oriented Line Formation problem starts from an arbitrary configuration with diameter $D$ where robots are on distinct points and has a goal of arranging robots in consecutive grid points on an oriented line, where the two end robots are distinguished by different light colors. The Oriented Line to Convex Hull problem (also called OLCH) starts with robots in consecutive grid points on an oriented line and has a goal of arranging robots on the corners of a convex hull. We previously developed [14] a randomized Oriented Line Formation algorithm that runs in $O(\max\{n, D\})$ time with high probability. (We sketch this algorithm in Section 5.4.) This algorithm uses randomization only for Leader Election. All subsequent algorithms use this Oriented Line Formation algorithm and develop ideas to solve OLCH.

- **Result 1**: We prove time and perimeter lower bounds of $\Omega(n^3)$, as well as an area lower bound of $\Omega(n^3)$, for any Convex Hull Formation algorithm (as well as any algorithm for the subproblem of OLCH) on a grid by autonomous robots with or without lights.

- **Result 2**: We present a generic algorithm framework that can be adapted on the basis of various input parameters. With our parameters, this template achieves optimal area, perimeter, and time with various results in terms of local computation.
Table 5.1. Asynchronous OLCH for \(n\) robots on an infinite grid. TT is short for Ternary Tree. The upper and lower bounds on time, perimeter, and area in the table can be extended to Convex Hull Formation by adding \(D\) to the upper and lower bounds to account for the case of a large initial configuration, e.g. \(D = \omega(n^2)\). Nevertheless, the optimalities are unchanged.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (epochs)</th>
<th>Perimeter (unit distance)</th>
<th>Area (unit distance(^2))</th>
<th>Local Computation</th>
<th>Number of Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>(\Omega(n^3))</td>
<td>(\Omega(n^3))</td>
<td>(\Omega(n^3))</td>
<td>(\Omega(1))</td>
<td>–</td>
</tr>
<tr>
<td>Quadratic</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
<td>(O(n^3))</td>
<td>(O(1))</td>
<td>30</td>
</tr>
<tr>
<td>TT</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
<td>(O(n^3))</td>
<td>(O(n \log n))</td>
<td>35</td>
</tr>
<tr>
<td>Improved TT</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
<td>(O(n^3))</td>
<td>(O(n))</td>
<td>35</td>
</tr>
<tr>
<td>Farey Seq.</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
<td>(O(n^3))</td>
<td>(O(n \log n))</td>
<td>35</td>
</tr>
</tbody>
</table>

- **Result 2a:** A deterministic, asynchronous OLCH algorithm with time \(O(n^2)\), perimeter \(O(n^2)\), and area \(O(n^3)\) using 30 colors requiring \(O(1)\) local computation. This leads to a randomized Convex Hull Formation algorithm with time \(O(\max\{n^2, D\})\) with high probability.

- **Result 2b:** A deterministic, asynchronous OLCH algorithm with time \(O(n^3)\), perimeter \(O(n^3)\), and area \(O(n^3)\) using 35 colors requiring \(O(n \log n)\) local computation. This leads to a randomized Convex Hull Formation algorithm with time \(O(\max\{n^3, D\})\) with high probability.

- **Result 2c:** Two more deterministic, asynchronous OLCH algorithms with time \(O(n^3)\), perimeter \(O(n^3)\), and area \(O(n^3)\) using 35 colors built off result 2b, but with improved local computation. The two algorithms require \(O(n)\) and \(O(n^2 \log n)\) local computation, respectively.

Results 2b and 2c have (asymptotically) optimal perimeter, time, and area complexities (from Result 1).

This chapter expands on the previous work by adding new metrics such as the area and local computation, and by adding new algorithms for Convex Hull Formation that improve upon the performance of previous results.

### 5.2. Previous Work

The most closely related previous work is that of Adhikary et al. [7] solving Complete Visibility (or Mutual Visibility) on a grid-based domain using asynchronous...
robots with lights with 11 colors. However, they established only that their algorithm finishes in finite time with no analysis of perimeter. Sharma et al. [14] provided an improved randomized algorithm that solves **COMPLETE VISIBILITY** on a grid-based domain using asynchronous robots with lights in time $O(\max\{D, n\})$ with 32 colors, where $D$ is the diameter of the initial configuration. Their algorithm first solves the **ORIENTED LINE FORMATION** problem, the result of which we use in this chapter, and then the **COMPLETE VISIBILITY** problem. **ORIENTED LINE FORMATION** can be solved using 29 colors and 3 additional colors to reach a complete visibility configuration starting from the **ORIENTED LINE FORMATION** configuration.

As noted in Chapter 3, several recent works provided upper bounds on time and the number of colors. None of these results consider the perimeter metric. Recently, Hector and Vaidyanathan [69] studied the distance and spatial complexities of these **CONVEX HULL FORMATION** algorithms and established certain cases to be optimal. The distance complexity is a reflection of the total distance moved by robots; this complexity is somewhat reflected in the time complexity in the model considered in this chapter as each robot is allowed to move only one unit of distance in an LCM cycle. The spatial complexity is a reflection of the amount of area (space) needed for the algorithm to run. This, too, is partially captured in the perimeter complexity of our results.

### 5.3. Chapter Roadmap

We discuss preliminaries in Section 5.4 and Result 1 (lower bounds) in Section 5.5. Section 5.6 sketches the ideas behind algorithms. Section 5.7 outlines the coprimes algorithm framework. Result 2a (Algorithm 1) is described in Section 5.8. Result 2b (Algo-
Algorithm 2) is explained in Section 5.9, and Result 2c (Algorithms 3 and 4) in Section 5.9.2. We then conclude in Section 5.11 with a short discussion.

5.4. Preliminaries

In this section, we formally define the Oriented Line Formation and OLCH algorithms. These combine to form a complete Convex Hull Formation algorithm.

5.4.1. Oriented Line Formation

The line formation algorithm of [14] first elects four leaders to serve as beacons and places them at corners of an axis-aligned rectangle such that the remaining $n - 4$ robots are in the interior of this rectangle. Two of these leaders define the target row in which the robots will form a line. Next, in each column, the robot closest to the target row moves to the target row if the point is unoccupied at the intersection of its column and that row. Following this, the remaining robots pipeline down their column to the row above the target row, then these columns of robots pipeline themselves along the row above the target row, dropping into vacant points on the target row. Finally, the robots compact themselves to be in consecutive positions. Note that the robots at the left and right ends of the configuration have distinct light colors, marking them as the left and right ends. For $n$ robots in an initial configuration with diameter $D$, this randomized algorithm solves Oriented Line Formation in $O(\max\{n, D\})$ time with high probability, using randomization for Leader Election. Further, since the randomized Leader Election algorithm selects a leader from among a constant number of robots, it terminates with probability $\geq 1 - \frac{1}{2^{\max\{n, D\}}}$. Furthermore, the randomized algorithm needs 29 colors to solve Oriented Line Formation [14]. In subsequent discussion, we only con-
sider cases where \( D \) is \( O(n^{\frac{3}{2}}) \), because obviously if \( D \) becomes very large, any algorithm for positioning robots on a line of consecutive grid points first must gather the robots sufficiently close.

5.4.2. Oriented Line to Convex Hull

After solving Oriented Line Formation, the robots in our algorithms solve OLCH as defined in Definition 5. These two problems combine to form an algorithm for Convex Hull Formation. To solve OLCH, we provide a purely deterministic algorithm. Thus, if robots are provided Leader Election (or some form of symmetry breaking to allow for it), Oriented Line Formation becomes deterministic as well, and therefore, all of Convex Hull Formation.

Definition 5 Let robots begin in consecutive positions along a line on the grid. Further, let the leftmost and rightmost robots on the line have unique, distinguishing colors. The problem of moving robots from this initial configuration to the corners of a convex polygon is called OLCH. \( \square \)

5.5. Lower Bounds

We prove the following theorem for the lower bound on both time and perimeter for any Convex Hull Formation algorithm (deterministic or randomized). We then provide a theorem for the lower bound on achievable area.

Theorem 5.1 For an initial configuration with robots on distinct grid points, any (deterministic/randomized) algorithm for Convex Hull Formation in the classical model with no lights or the robots with lights model needs \( \Omega(n^{\frac{3}{2}}) \) rounds in the FSYNC setting. Moreover, the perimeter of a solution to Convex Hull Formation must be \( \Omega(n^{\frac{3}{2}}) \).
Proof: We first provide the lower bound of $\Omega(n^{\frac{3}{2}})$ on the perimeter of a solution to Convex Hull Formation. We use the result due to Balog and Barany [70] which says the following: Let $H(r)$ denote the convex hull of the grid points that are on or in the interior of the circle of radius $r$, the number of vertices $N(r)$ of $H(r)$ satisfies $c_1 r^{\frac{3}{2}} \leq N(r) \leq c_2 r^{\frac{3}{4}}$ for large enough $r$, where $c_1$ and $c_2$ are absolute constants. Setting $r = \Theta(n^{\frac{3}{2}})$, we have $c_1 n \leq N(n^{\frac{3}{2}}) \leq c_2 n$. For a circle of radius $\Theta(n^{\frac{3}{2}})$, the perimeter is also $\Theta(n^{\frac{3}{2}})$.

We now provide the lower bound of $\Omega(n^{\frac{3}{2}})$ on the time of any algorithm for Convex Hull Formation. We prove this bound in the $\mathcal{FSYNC}$ setting without lights, which applies immediately to the $\mathcal{ASYNC}$ setting as well as the robots with lights model. The lower bound construction is as follows. Suppose $\sqrt{n} \geq 4$ is an even integer. Consider a $\sqrt{n} \times \sqrt{n}$ axis-aligned sub-grid $G'$ having $n$ grid points with perimeter $P'$. Let $G'$ be positioned in the middle of a circle with radius $\Theta(n^{\frac{3}{2}})$ having $\Theta(n^3)$ grid points within the infinite grid and call this grid as $G$. Let $P$ be the perimeter of $G$. The distance $L$ from any grid point of $G'$ on its perimeter $P'$ to the closest grid point of $G$ on its perimeter $P$ is at least $\Theta(n^{\frac{3}{2}}) - \sqrt{\frac{P}{2}}$. Suppose $n$ robots are positioned initially on the $n$ grid points of $G'$, one on each grid point. We show that at least one robot in $G'$ needs to move $\Omega(n^{\frac{3}{2}})$ times, giving the $\Omega(n^{\frac{3}{2}})$ rounds lower bound on time in the $\mathcal{FSYNC}$ setting.

We have from the perimeter lower bound that $G$ must have the perimeter of $\Theta(n^{\frac{3}{2}})$. Therefore, any robot in $G'$ needs to move at least $\Theta(n^{\frac{3}{2}}) - \sqrt{\frac{P}{2}} = \Omega(n^{\frac{3}{2}})$ rounds to reach its position on $P$, for any even $\sqrt{n} \geq 4$. Note also that the $n$ robots in $G'$ do not need to move to the nodes of $G$ as described above, but if they move otherwise it will only increase the minimum number of rounds required for at least one robot.

\textbf{Theorem 5.2} For an initial configuration with robots on distinct grid points, any algo-
Proof: We use the result of Cai et al. [71] which proves that the least possible area $a(n)$ of a convex lattice polygon with $n$ vertices satisfies $a(n) \geq \frac{1}{1152}n^3 + O(n^2)$. This immediately applies to any algorithm for Convex Hull Formation since the goal is to create a convex polygon with $n$ vertices on the integer lattice (grid).

**Theorem 5.3** Any algorithm for OLCH in any model requires $\Omega(n^{\frac{3}{2}})$ time, $\Omega(n^{\frac{3}{2}})$ perimeter, and $\Omega(n^3)$ area.

Proof: The lower bound on perimeter follows from the proof of Theorem 5.1, since the lower bound on perimeter holds for any initial configuration. The lower bound on time is similar to the proof of Theorem 5.1, with the exception that the initial configuration consists of $n$ robots lying in consecutive positions on a line rather than in a $\sqrt{n} \times \sqrt{n}$ axis-aligned sub-grid. Nevertheless, $\Theta(n^{\frac{3}{2}}) - n$ is still $\Theta(n^{\frac{3}{2}})$; this means that one robot must still travel $\Omega(n^{\frac{3}{2}})$ times, giving the $\Omega(n^{\frac{3}{2}})$ rounds lower bound on time in the $\mathcal{FSYNC}$ setting. Since $\mathcal{FSYNC}$ is a special case of $\mathcal{SSYNC}$ and $\mathcal{ASYNC}$, the lower bound on time immediately applies to these as well. Finally, the area lower bound follows from the proof of 5.2, since the lower bound on area holds for any initial configuration.

**5.6. Coprimes Strategy**

We now present a method of using coprime integer pairs to solve Convex Hull Formation. We later discuss methods for generating these coprime pairs and their trade-offs.
5.6.1. Coprime Pairs

Before considering **Convex Hull Formation**, a few definitions and results are needed.

**Definition 6** For any positive integers \(a, b, c, d\), let \((a, b)\) and \((c, d)\) be ordered pairs. The slope of \((a, b)\) is defined to be \(\frac{b}{a}\). We say that \((a, b) = (c, d)\) iff \(a = c\) and \(b = d\).

**Definition 7** An ordered pair of positive integers \((a, b)\) is called a coprime pair iff \(a \neq b\) and the only common factor of both \(a\) and \(b\) is 1.

Observe that \(\gcd(a, b) = 1\). In general, we will write a coprime pair \((a, b)\) with \(a < b\).

**Lemma 5.4** Every coprime pair \((a, b)\) has a unique slope \(\frac{b}{a}\).

**Proof:** Suppose, by contradiction, there exist coprime pairs \((c, d) \neq (a, b)\) such that \(\frac{d}{c} = \frac{b}{a}\). If \(a = c\), then \(b = d\), and \((a, b) = (c, d)\). Without loss of generality, let \(a < c = \alpha a\), for some \(\alpha > 1\). Then \(d = \alpha b\). If we can show that \(\alpha\) is an integer, then \((c, d)\) will not be a coprime pair, providing the necessary contradiction. Observe that \(c = d \left(\frac{a}{b}\right) = a \left(\frac{d}{b}\right)\). Since \((a, b)\) is a coprime pair and \(c\) is an integer, \(\frac{d}{b} = \alpha\) must be an integer, providing the necessary contradiction.

Lemma 5.4 establishes that if we generate \(n\) coprime pairs, then we would have generated \(n\) distinct slopes.

**5.6.2. Using Unique Slopes to Form a Convex Hull**

Let \(\mathbb{Z}^+ = \{1, 2, \cdots\}\) be the set of positive integers and let \(\mathbb{N} = \{0, 1, 2 \cdots\}\) be the set of non-negative integers. Let \(f, g : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+\) be two functions with properties specified below. For any \(i \geq 1\), define the \(i^{th}\) slope to be the real number \(S(i) = \frac{g(i)}{f(i)}\). Let
Define a second pair of functions $F, G : \mathbb{Z}^+ \rightarrow \mathbb{N}$ as follows:

$$F(1) = 0 \quad \text{and} \quad G(1) = 0 \quad (5.2)$$

$$\forall i > 1, \quad F(i) = \sum_{k=1}^{i} f(k) \quad \text{and} \quad G(i) = \sum_{k=1}^{i} g(k) \quad (5.3)$$

Observe that $F, G$ are increasing functions for $i \geq 1$.

An algorithm that positions points at $(F(i), G(i))$ will form a profile of “increasing slopes” $S(i)$, as shown in Figure 5.1. Clearly, they will be positioned at the corners of a convex hull as required for OLCH. Key to this approach is the increasing slopes given by Equation (5.1).

The time complexity of the algorithm depends on the algorithmic details. Let
Peri($S$) and Area($S$) represent the perimeter and area of the smallest enclosing rectangle around the positions ($F(i), G(i)$) for $1 \leq i \leq n$, respectively. The perimeter of the smallest enclosing rectangle around the convex hull generated by the algorithm is then

$$Peri(S) = 2(F(n) - F(1) + G(n) - G(1))$$

$$= O(F(n) + G(n))$$

(5.4)

Similarly, the area is given by

$$Area(S) = ((F(n) - F(1))(G(n) - G(1))$$

$$= O(F(n)G(n))$$

(5.5)

Thus, by generating $n$ coprime pairs and ordering them by slopes, we have a sequence of offsets between points on the integer plane that form a convex hull. In the following section, we explain and evaluate various algorithms for generating these coprime pairs and performing the proposed OLCH algorithm.

5.7. Coprimes Algorithm Framework

In this section, we provide a framework for solving OLCH. The input is a set of slopes such that, given $n$ and $i$, the $i^{th}$ coprime pair $(a_i, b_i)$ can be determined.

Our strategy is to use two special robots called $n$-beacons to move a distance $d = n - 2$ apart from each other so that robots executing the algorithm all know the value of $d$, the number of non-beacon robots, and will use context to determine their respective values of $i$. A robot will say it is the $i^{th}$ robot if it is initially a horizontal distance $i$ away from the $n$-beacons. These two $n$-beacons are selected as the two leftmost robots along the initial line. The leftmost robot $r_1$ colors itself $n$Top and moves up a distance $d - 1$ and
Table 5.2. Coprime Framework Algorithm Summary

<table>
<thead>
<tr>
<th>Step</th>
<th>Movement of Robots</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Beacon robots move into position</td>
</tr>
<tr>
<td>2</td>
<td>Non-beacon robots move up and right</td>
</tr>
<tr>
<td>3</td>
<td>Non-beacon robots move back left</td>
</tr>
<tr>
<td>4</td>
<td>The lower beacon moves into position</td>
</tr>
<tr>
<td>5</td>
<td>The upper beacon moves into position</td>
</tr>
</tbody>
</table>

moves right to align itself vertically with the other n-beacon, which is the second leftmost robot. Subsequently, the second leftmost robot $r_b$ colors itself $n_{Bottom}$ and moves down by 1 unit. This movement is summarized in Figure 5.2. From this position, all non-beacon robots on the line can see $n_{Top}$ and $n_{Bottom}$, and thus, know $n$ as shown in Figure 5.3.

The algorithms under this framework are outlined as shown in Table 5.2. The steps proceed sequentially with no overlap between steps. The broad strategy of the algorithm is as follows. After the beacons are positioned (Step 1), the non-beacon robots $r_i \; \forall \; 1 \leq i \leq d$ move to their target positions. The non-beacons move to their target positions by first moving vertically to row $y'_{i}$, the destination
Figure 5.3. Determining $n$ And $i$ From the Beacons

row. They then move right to column $x'_d$, the target column of the rightmost robot in the initial configuration on the line (completing Step 2). Finally, each robot moves left along its current row to its target column $x'_i$ (Step 3). Figure 5.4 shows the general movements of the non-beacon robots corresponding to Steps 2 and 3 of the algorithm. At the end of Step 3, each non-beacon robot $r_i$ has reached its final target position $(x'_i, y'_i)$. Then, the robot $r_b$ moves vertically then horizontal towards $(x'_b - c_1, y'_b)$ where $c_1$ is a small, positive integer (Step 4). Similarly, $r_t$ will then move to $(x'_t, y'_t + c_2)$ (Step 5).

All robots, except the two $n$-beacons, will initially be colored moveUp. Each robot will proceed through one of the following color patterns throughout the algorithm’s execution.

The discussion below refers to the directions left and right and to particular coordinates, and these are global directions and coordinates. Recall, however, that robots do not have a common orientation. The conditions and actions assume that a robot sees one or
both beacons to orient itself and identify the global coordinate system. If a robot cannot
orient itself, then it does not satisfy a condition and it takes no action.

5.7.1. Moving Up

The idea is that robots move upward in a line asynchronously, so the robots with
color \texttt{moveUp} will be in one or two rows at any time. As a robot reaches its goal row, it
changes color to \texttt{moveRight}.

Robots colored \texttt{moveUp} execute the following condition-action pairs.

\textbf{Condition U1:} Robot $r_i$ sees a non-beacon robot to its left and above it.

\textbf{Action U1:} Robot $r_i$ moves upward by one.

\textbf{Condition U2:} Robot $r_i$ sees the \textit{n-beacons} and determines it is at its goal height.

\textbf{Action U2:} Robot $r_i$ changes its color to \texttt{moveRight}.

---

### Table 5.3. Coprime Framework Algorithm Color Sequences

<table>
<thead>
<tr>
<th>Robot</th>
<th>Color Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leftmost beacon $r_t$</td>
<td>nTop $\rightarrow$ done</td>
</tr>
<tr>
<td>Other beacon $r_b$</td>
<td>nBottom $\rightarrow$ done</td>
</tr>
<tr>
<td>Rightmost robot</td>
<td>moveUp $\rightarrow$ moveRight $\rightarrow$ moveLeft $\rightarrow$ done</td>
</tr>
<tr>
<td>All others</td>
<td>moveUp $\rightarrow$ moveRight $\rightarrow$ done</td>
</tr>
</tbody>
</table>
Condition U3: Robot $r_i$ sees the $n$-beacons, determines it is not at its goal height, and there are no robots colored moveUp below it.

Action U3: Robot $r_i$ moves upward by one.

5.7.2. Moving Right

The idea is as follows. Each robot with color moveRight is on its goal row. They move to the right until reaching column $x_d'$. There will be a front line of moveRight robots, but not necessarily all moveRight robots will be in that line because of activation schedules. When all moveRight robots are on row $x_d'$, the top robot is in its destination point and changes color to done, then the other moveRight robots change color to moveLeft.

Robots colored moveRight execute the following condition-action pairs.

Condition R1: Robot $r_i$ sees only the beacon robots and one non-beacon robot that is directly below it and colored moveRight, and it has reached $(x_d', y_i')$.

Action R1: Robot $r_i$ changes its color to done and terminates.

Condition R2: Robot $r_i$ sees a robot colored done or moveLeft.

Action R2: Robot $r_i$ changes its color to moveLeft.

Condition R3: Robot $r_i$ sees a robot to its right colored moveRight.

Action R3: Robot $r_i$ moves to the right by one.

Condition R4: Robot $r_i$ has not reached $(x_d', y_i')$, all non-beacon robots at no greater height than $r_i$ are colored moveRight, $r_i$ sees no robots to its left colored moveRight, and there does not exist a non-beacon robot $r_j$ such that $y_j > y_i$ and $x_j \leq x_i$. 

52
Action R4: Robot $r_i$ moves to the right by one.

5.7.3. Moving Left

The idea is that robots move leftward in a line asynchronously, so the robots with color moveLeft will be in one or two columns at a time. As a robot reaches its destination point, it changes color to done.

Robots colored moveLeft execute the following condition-action pairs.

Condition L1: Robot $r_i$ is at $(x'_i, y'_i)$ and does not see a robot to its right colored moveLeft.

Action L1: Robot $r_i$ changes its color to done and terminates.

Condition L2: Robot $r_i$ does not see a robot at a greater height colored moveLeft or to its right colored moveLeft.

Action L2: Robot $r_i$ moves to the left by one.

Condition L3: Robot $r_i$ sees a robot above it and to the left colored moveLeft.

Action L3: Robot $r_i$ moves to the left by one.

5.7.4. Move Bottom Beacon

Once all robots other than the beacons are in their destination points, the bottom beacon moves to its destination point.

The robot colored nBottom executes the following condition-action pairs.

Condition B1: Robot $r_i$ is at position $(x'_1 - C, y'_1)$.

Action B1: Robot $r_i$ colors itself done and terminates.

Condition B2: Robot $r_i$ sees only robots colored done or nTop.
Action B2: Robot $r_i$ moves towards $(x'_i - C, y'_i)$, vertically then horizontally as needed.

5.7.5. Move Top Beacon

An astute reader will notice that a convex hull has already been achieved, and thus, complete visibility. This is due to the fact that all robots other than $r_t$, the robot colored $n_{Top}$, form an arc offering mutual visibility to all other robots within the arc. The robot $r_i$ is at $(x_i, y_i)$ such that $x_i < x_i \forall i$, $y_i > y_1$, and the arc is an increasing curve, and thus, no robot can block $r_i$’s view of the remaining robots, or vice versa. Nevertheless, it may be desirable for the algorithm to terminate with all robots in a state of complete visibility on the arc. Therefore, the robot colored $n_{Top}$ executes the following condition-action pairs.

**Condition T1:** Robot $r_i$ is at position $(x'_d, y'_d + C)$.

**Action T1:** Robot $r_i$ colors itself *done* and terminates.

**Condition T2:** Robot $r_i$ only sees robots colored *done*.

**Action T2:** Robot $r_i$ moves towards $(x'_d, y'_d + C)$, vertically then horizontally as needed.

5.7.6. Analysis of Coprimes Framework

**Lemma 5.5** All robots with color *moveUp* will be in one or two rows at a time.

**Proof:** Let all robots with color *moveUp* be in two rows $k$ and $k - 1$. We will establish that no robot will move up from row $k$ to row $k + 1$ (until all robots with color *moveUp* are in row $k$). Condition U2 does not result in moving a robot, and Condition U3 is not satisfied by robots in row $k$. Condition U1 is the only possibility by which a robot $r_i$ in row $k$ with color *moveUp* could move to row $k + 1$. Robots with color *moveRight* get that color by Action U2 and stay on the same row and robots with color *moveLeft* were previously
colored \texttt{moveRight} and they stay on the same row, so these cannot be higher than row \(k\). Consequently, Condition U1 is not satisfied for any robot with color \texttt{moveUp} on row \(k\), and no robot with color \texttt{moveUp} moves to row \(k+1\) while a robot with color \texttt{moveUp} is on row \(k-1\).

**Lemma 5.6** When robot \(r_i\) changes color from \texttt{moveUp} to \texttt{moveRight}, (i) each \(r_j, 1 \leq j < i-1\), to the left has already changed color to \texttt{moveRight}, and (ii) \(r_{i-1}\) either has already changed color to \texttt{moveRight} or it will change color to \texttt{moveRight} in the next epoch.

**Proof:** When \(r_i\) changes color to \texttt{moveRight} in Action U2, it is on its destination row \(y'_i\). (i) The destination row \(y'_j\) of each \(r_j, 1 \leq j < i-1\), is at least two rows lower than \(y'_i\), so by Lemma 5.5, \(r_j\) must have already changed color to \texttt{moveRight}. (ii) The destination row \(y'_{i-1}\) of \(r_{i-1}\) could be one row below \(y'_i\). In that case, \(r_{i-1}\) must be on row \(y'_{i-1}\) by Lemma 5.5. If \(r_{i-1}\) does not already have color \texttt{moveRight}, then within one epoch, \(r_{i-1}\) will satisfy Condition U2 and change color to \texttt{moveRight}.

**Lemma 5.7** Until all robots are in column \(x'_d\), either or both of the following happens each \(O(1)\) epochs: (i) the bottommost row of robots with color \texttt{moveUp} will move up one row, or (ii) the leftmost column of robots with color \texttt{moveRight} will move right one column.

**Proof:** Consider the bottommost row of robots with color \texttt{moveUp}. If one of these robots (the leftmost one) is at its goal height and sees the n-beacons, then it changes its color to \texttt{moveRight}. The other robots on this row are not yet at their goal heights. By Lemma 5.5, either this will be the only row of robots with color \texttt{moveUp} or this will be the lower of two rows of robots with color \texttt{moveUp}. In the former case, if a robot with color \texttt{moveUp} sees both beacons, then it will move up one row. In the latter case, if a robot \(r_i\), in the
bottommost row sees a robot in the higher row with color \texttt{moveUp} to its left (Condition U1), then \( r_i \) will move up one row (Action U1). Otherwise, if a robot \( r_i \) in the bottommost row does not see a robot in the higher row to its left, but it does see both beacons (Condition U3), then \( r_i \) will move up one row (Action U3). In these circumstances, the bottommost row of robots with color \texttt{moveUp} will move up by one row in \( O(1) \) epochs.

If Conditions U1-U3 are not satisfied for some robot \( r_i \) with color \texttt{moveUp}, then \( r_i \) must see the top beacon (because Condition U1 is not satisfied) and its view of the bottom beacon must be blocked (because Conditions U2 and U3 are not satisfied). The robots blocking views of the bottom beacon will have color \texttt{moveRight} (Lemma 5.6), and at least one of these robots must be to the left of the robot with color \texttt{moveUp}. (Note: By Lemma 5.6(ii), one blocking robot might have color \texttt{moveUp}, but that robot will change color to \texttt{moveRight} in the next epoch.) In this circumstance, consider the leftmost column of robots with color \texttt{moveRight}. If one or more robots with color \texttt{moveRight} are to the right of this column (Condition R3), then all of the robots in the leftmost column will see such a robot and move right by one column (Action R3). Otherwise, the topmost robot in this column of robots with color \texttt{moveRight} will satisfy Condition R4 and move right (Action R4). In the next epoch, the remaining robots in the leftmost column will see a robot to their right with color \texttt{moveRight} and move right themselves. So, if not all the robots in the bottommost row of robots with color \texttt{moveUp} move up one row in \( O(1) \) epochs, then all robots in the leftmost column of robots with color \texttt{moveRight} will move right in \( O(1) \) epochs.

\[ \square \]

**Corollary 5.8** Each robot \( r_i \) with color \texttt{moveUp} will reach its destination row \( y'_i \) within \( O(G(n)) \) epochs.
Corollary 5.9 Each robot $r_i$ with color \texttt{moveRight} will reach column $x'_d$ at point $(x'_d, y'_i)$ within $O(F(n))$ epochs. □

The following lemma holds by a proof similar to that of Lemma 5.5.

Lemma 5.10 All robots with color \texttt{moveLeft} will be in one or two columns at a time. □

Lemma 5.11 All robots colored \texttt{moveLeft} will reach their final destination $(x'_i, y'_i)$ within $O(F(n))$ epochs.

Proof: When all robots are in column $x'_d$ with color \texttt{moveRight}, the topmost robot will satisfy Condition R1 and change its color to \texttt{done} (Action R1). Then the robot below this one satisfies Condition R2 and changes its color to \texttt{moveLeft} (Action R2). This robot next satisfies Condition L2 and moves left by one (Action L2). In the next epoch, the remaining robots with color \texttt{moveRight} satisfy Condition R2 and change color to \texttt{moveLeft} (Action R2). So, within $O(1)$ epochs, all robots have changed color to \texttt{done} or \texttt{moveLeft}.

By Lemma 5.10, all robots with color \texttt{moveLeft} are in one or two columns. We now establish that within $O(1)$ epochs, the rightmost column of robots with color \texttt{moveLeft} will move left by one column (or change color to \texttt{done}). If a robot $r_i$ in the rightmost column is at its destination point $(x'_i, y'_i)$ (Condition L1), then it will change color to \texttt{done} (Action L1). So we can focus on only the robots colored \texttt{moveLeft} that are not at their destination points. Consider the rightmost column of robots with color \texttt{moveLeft}. If the robots with color \texttt{moveLeft} are in two columns, then the robots in the rightmost column will satisfy condition L2 or L3 and move left by one column (Action L2 or L3). Now consider the case in which robots with color \texttt{moveLeft} are in one column. The topmost robot with color \texttt{moveLeft} in this column will satisfy Condition L2, so it will move left by one column (Action L2). In the next epoch, the remaining robots with color \texttt{moveLeft} in that
rightmost column will see that topmost robot and satisfy Condition L3, so they will move left by one column (Action L3).

So, every $O(1)$ epochs, the rightmost column of robots with color moveLeft will move to the left by one column, and when a robot reaches its destination point, it will change color to done and terminate. As the robots start from column $x'_d$, the distance to the destination point for each robot is $O(F(n))$, so in $O(F(n))$ epochs, all robots will reach their destination points.

□

**Lemma 5.12** After all non-beacon robots have color done, the robot colored nBottom will reach its final position within $O(1)$ epochs.

**Proof:** By Lemma 5.10 and Condition L1, the final robot to change its color from moveLeft to done will be $r_1$, the bottom-most robot that is not an $n$-beacon. Therefore, no robot will block $r_1$ from robot $r_b$, and when $r_1$ colors itself done, the remaining non-beacons will be done as well. At this point, $r_b$ is a constant distance from $r_1$. Robot $r_b$ then satisfies Condition B2 and begins moving towards a goal point that is a small, positive integer $C$ offset from $(x'_1, y'_1)$ (Action B2). Since $x'_1 = 0$ and $d > 1$ for $n > 3$ we know that $r_b$ will not collide with the other beacon (colored nTop and located at height $d - 1$) as $r_b$ moves vertically to height $y'_1$. Then $r_b$ moves horizontally towards $x'_1 - C$ unimpeded by any robots along its path as no robots other than $r_b$ and $r_1$ at $(x'_1, y'_1)$ will be at this height.

Because $r_b$ does not have to wait once it starts moving, it will take $O(1)$ epochs to reach its goal point and terminate.

□

**Lemma 5.13** After all non-beacon robots and the bottom beacon have color done, the robot colored nTop will reach its final position within $O(F(n) + G(n))$ epochs.
Proof: Before the \( n \)-beacon robot colored \( n\text{Bottom} \) changes its color to \textbf{done}, robot \( r_t \) colored \( n\text{Top} \) can always see a robot colored \( n\text{Bottom} \) because they are both positioned to the left of all other robots. After the bottom beacon changes its color to \textbf{done}, \( r_t \) satisfies Condition T2 and begins moving towards a goal point that is a small, positive integer \( C \) offset from \( (x'_d, y'_d) \). It is unimpeded by any robots along its path because it will first move to height \( y'_d + C \) before moving horizontally. Robot \( r_t \) starts at point \((-1, n - 3)\) and \( x'_d = O(F(n)) \) and \( y'_d = O(G(n)) \), so \( r_t \) will reach its goal point and terminate in \( O(F(n) + G(n)) \) epochs, completing the algorithm.

Theorem 5.14 Any algorithm under the coprimes algorithm framework for solving OLCH runs in \( O(F(n) + G(n)) \) time and achieves \( O(F(n) + G(n)) \) perimeter and \( O(F(n)G(n)) \) area on a robots with lights model with 35 colors in \texttt{ASYNC}.

Proof: The time bound comes from Corollaries 5.8 and 5.9 and Lemmas 5.11, 5.12, and 5.13. The color count comes from the 6 used in the proposed algorithm framework in addition to the 29 colors used in the oriented line formation on a grid [14].

We now detail various algorithms for using \( i \), and sometimes \( n \), to position robots on corners of the convex hull starting from an oriented line. We also describe how robots determine the values of \( n \) and \( i \). The algorithms, and their performance metrics, differ based on the list of coprime pairs.

5.8. Quadratic Algorithm

We present here a simple asynchronous algorithm that solves OLCH in \( O(n^2) \) epochs and with \( O(n^2) \) perimeter. We express the algorithm within the framework described in Section 5.6. However, this algorithm is even simpler than described in the
framework, as we will soon see. This is because no robots need to determine the value of $n$ at any point. Therefore, we do not use beacon robots for the following algorithm. The remainder of the framework holds, and we prove that the algorithm still solves OLCH even without knowledge of $n$.

Recall that robots $r_1, r_2, \cdots, r_n$ begin OLCH in adjacent grid points on a line such that $r_i$ is at position $(i - 1, 0)$. Let $L_{OR}$ denote the line along which these robots are located. We let $F(1) = f(1) = 0$ and $G(1) = g(1) = 0$, meaning that the first of our $n$ pairs is $(0, 0)$. That is, robot $r_1$ begins in its final position. Let $f(i) = 1$ for $i > 1$ and $g(i) = i - 1$ for $i \geq 1$. Consequently, $F(i) = i - 1$ for $i \geq 1$, and $G(i) = \sum_{k=1}^{i} k - 1 = \frac{i(i - 1)}{2}$.

It can be verified that the function satisfies the requirement of Equation (5.1). Thus the goal of the algorithm is to move the robot initially at position $(i - 1, 0)$, along the $y$ direction to $(i - 1, \frac{i(i - 1)}{2})$.
The algorithm proceeds as follows. Robot $r_2$ is the first to move. It moves to height 1 from $L_{OR}$. Without loss of generality, let this point be $(1, 1)$. Robot $r_2$ now changes its color to \textbf{final} from \textbf{middle}. All the robots on $L_{OR}$ can now see $r_2$. Let the line passing through $(1, 1)$, where $r_2$ is currently positioned (after its move), that is parallel to $L_{OR}$ be called $L^1_{OR}$. In a similar way, call the line parallel to $L_{OR}$ that passes through $(0, i)$ as $L^i_{OR}$. After $r_2$ is colored \textbf{final}, all remaining robots $r_3, \ldots, r_n$ on $L_{OR}$ move to be positioned on $L^1_{OR}$. We synchronize the moves in such a way that no robots on $L^1_{OR}$ move to line $L^2_{OR}$ until all robots on $L_{OR}$ have moved to $L^1_{OR}$. This is crucial in making the algorithm work in the \textbf{ASYNC} setting. After all robots $r_2, \ldots, r_n$ move to $L^1_{OR}$, $r_2$ stays on $L^1_{OR}$ and robots $r_3, \ldots, r_n$ next move to $L^2_{OR}$. After this, robots $r_3, \ldots, r_n$ move to $L^3_{OR}$. Robot $r_3$ now changes its color to \textbf{final}. This process then repeats until $r_4, \ldots, r_n$ reach $L^6_{OR}$. Robot $r_4$ stays on $L^6_{OR}$ with a color of \textbf{final} and robots $r_5, \ldots, r_n$ move to $L^7_{OR}$. After these robots reach $L^{10}_{OR}$, $r_5$ stays on $L^{10}_{OR}$ and robots $r_6, \ldots, r_n$ move to $L^{11}_{OR}$. Running the algorithm this way, robot $r_i$ stays on the grid point on line $L^j_{OR}$, where $j = G(i) = \sum_{k=1}^{i} k - 1$. After robot $r_n$ reaches line $L^j_{OR}$ and stays there, the algorithm finishes solving OLCH.

**Lemma 5.15** The quadratic algorithm correctly solves OLCH without the use of beacon robots.

**Proof:** Notice that $r_i, i \leq 1$, moves on a line perpendicular to $L_{OR}$ through its starting position, and $r_i$ never leaves this line. The robots that have not reached their final positions move from $L^j_{OR}$ to $L^{j+1}_{OR}$ at all times since below line $L^j_{OR}$ (towards the x axis), they only see \textbf{final} and \textbf{leader1} robots. This shows that the algorithm works in the asynchronous setting. No robot moves from $L^j_{OR}$ to $L^{j+1}_{OR}$ before all robots have moved from
Consider any three consecutive robots \( r_{i-1}, r_i, r_{i+1} \). Robot \( r_i \) comes to rest (or stays) on a grid point only after \( r_{i-1} \) stays and \( r_{i+1} \) stays on a grid point only after \( r_i \) stays. Also, \( r_i \) moves further from \( L_{OR} \) compared to \( r_{i-1} \). Consider the final positions of \( r_{i-1}, r_i, \) and \( r_{i+1} \) as described in the algorithm. Connect the final positions \( r_{i-1} \) and \( r_i \) by line \( L_1 \) and the final positions \( r_i \) and \( r_{i+1} \) by line \( L_2 \). The slope of line \( L_1 \) is 
\[
\left( \sum_{k=1}^{i}(k-1) \right) - \left( \sum_{k=1}^{i-1}(k-1) \right) = i - 1,
\]
since the x-coordinates of \( r_i \) and \( r_{i-1} \) differ only by 1. Similarly, the slope of line \( L_2 \) is 
\[
\left( \sum_{k=1}^{i+1}(k-1) \right) - \left( \sum_{k=1}^{i}(k-1) \right) = i.
\]
Again, the x-coordinates of \( r_{i+1} \) and \( r_i \) differ only by 1. Since the slopes of \( L_1 \) and \( L_2 \) differ, the final positions of any three consecutive robots \( r_{i-1}, r_i, r_{i+1} \) are not on a line. Moreover, since the increment in the \( y \) coordinate is \( g(i+1) - g(i) = 1 \) in going from \( r_i \) to \( r_{i+1} \), taking an angle \(< 180^\circ\) between \( L_1 \) and \( L_2 \) for all triples of consecutive robots provides a convex hull configuration. \( \square \)

**Theorem 5.16** The quadratic algorithm for OLCH runs in \( O(n^2) \) time and achieves \( O(n^2) \) perimeter.

**Proof:** By Theorem 5.14, any Coprimes algorithm under this framework for solving OLCH achieves \( O(F(n) + G(n)) \) time and perimeter. By design, \( F(n) = n - 1 \) and \( G(n) = \frac{n(n-1)}{2} \). Then \( O(n^2 + n - 1) = O(n^2) \) \( \square \)

**Theorem 5.17** The quadratic algorithm for OLCH achieves \( O(n^3) \) area.

**Proof:** Also by Theorem 5.14, any Coprimes algorithm under this framework for solving OLCH achieves \( O(F(n)G(n)) \) area. \( O(F(n)G(n)) = \left( (n-1) \left( \frac{n(n-1)}{2} \right) \right) = O(n^3) \) \( \square \)

**Theorem 5.18** The Oriented Line Formation algorithm of Sharma et al. [14] and the quadratic algorithm for OLCH solve Convex Hull Formation with 30 colors.
Proof: Starting from $L_{OR}$, the algorithm only needs 1 new color final. The other 29 colors were used in our previous line formation algorithm on a grid [14]. \hfill \Box

Before we end the discussion on the quadratic algorithm, we observe that since $f(k) = 1 \forall k > 1$ \((f(1) = 0)\) for the algorithm in this section, Equation (5.1) dictates that $g(k) < g(k + 1)$. The smallest value of $g(k + 1)$ that satisfies this constraint is $g(k + 1) = g(k) + 1 = k + 1$. Thus the quadratic algorithm is the best possible if $f(k) = 1$.

We also observe that if the real plane (rather than an integer plane) was used, then one could select $G(k)$ to be any non-linear monotonic function (even a very low growth real function such as $\log^{(x)} n = \log \log \cdots \log n$, for some fixed $x$). This clearly cannot be done on an integer plane where the monotonic function must produce an integer as its output.

5.9. Ternary Tree Algorithm

We now present an asynchronous algorithm that makes use of the value of $n$ to solve Oriented Line to Convex Hull in $O(n^2)$ epochs with $O(n^3)$ perimeter and $O(n^3)$ area. Figure 5.6 shows an example of solving OLCH using the Ternary Tree algorithm.

5.9.1. Generating Coprime Pairs and Ordering Them by Slope

The following well-known result is the basis for our method to generate coprime pairs.

**Lemma 5.19** [72] If $(a, b)$ is a coprime pair, then the pairs $(b, 2b - a)$, $(b, 2b + a)$, and $(a, b + 2a)$ are coprime pairs as well. \hfill \Box

Lemma 5.19 implies that one can recursively generate a ternary tree of “descendents” of $(a, b)$ by further expanding $(b, 2b - a)$ to coprime pairs $(2b - a, 3b - 2a)$, $(2b - a, 5b - 2a)$, and
Figure 5.6. Initial (black circles) and final locations (unshaded circles) of robots for an example of the Coprimes algorithm. For clarity, the axes are scaled differently. For this example, the coprime pairs (in order of their slopes) are \((4, 5), (3, 4), (2, 3), (1, 2), (1, 3), (1, 4)\). These coprime pairs are applied to robots \(r_i\) for \(1 \leq i \leq 6\). Robot \(r_0\) moves to a position consistent with the convex hull.

\((b, 4b - a)\), and so on. Let us call this ternary tree with \((a, b)\) as the root as the descendent tree of \((a, b)\). In particular, denote the descendent trees of coprime pairs \((1, 2)\) and \((1, 3)\) by \(T_2\) and \(T_3\), respectively.

**Lemma 5.20** \([72]\) Every coprime pair \((a, b)\) appears exactly once in either \(T_2\) or \(T_3\). \(\square\)

Lemma 5.20 implies that one can exhaustively generate all coprime pairs without duplicates by traversing the descendent trees \(T_2\) and \(T_3\).

The following observation is easy to see from Lemma 5.19.

**Lemma 5.21** If coprime pair \((x, y) \neq (a, b)\) exists in the descendent tree of \((a, b)\), then \(y > b\). \(\square\)

At this point it is worth recalling that the perimeter complexity of this approach is the following: \(O(F(n) + G(n)) = O\left(\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i\right)\) (from Equations 5.3 and 5.4). Thus an arbitrary set of \(n\) coprime pairs may not suffice for an optimal algorithm.
The next result provides a guide to selecting a good set of coprime pairs from descendent trees $T_2$ and $T_3$. There is also the question of how far we need to traverse $T_2$ and $T_3$ before a suitable set of $n$ coprime pairs is generated.

**Definition 8** [73] For any integer $m \geq 1$, Euler's totient function $\varphi(m)$ equals the number of positive integers $x$ (where $1 \leq x < m$) such that $(x, m)$ is a coprime pair. □

The following property of Euler's totient functions is key to our approach.

**Lemma 5.22** [74] As integer $m \rightarrow \infty$, the following holds:

$$
\sum_{i=1}^{m} \varphi(i) = \frac{3m^2}{\pi^2} + O\left(m(\log m)^{\frac{3}{2}}(\log \log m)^{\frac{4}{3}}\right).
$$

In other words, the sum of the first $m$ totient numbers is $\Theta(m^2)$. Clearly $\varphi(i) < i$.

The above lemma establishes that, averaged over a large number of values, $\varphi(i)$ is not very far from $i$; the sum of the first $m$ integers is also $\Theta(m^2)$.

If we set $m = \lceil \pi \sqrt{\frac{n}{3}} \rceil$ in Lemma 5.22, then for large $m$ we have $n < \sum_{i=1}^{m} \varphi(i) < 2n$ coprime pairs.

Thus the required number $n$ of coprime pairs can be generated by traversing $T_2$ and $T_3$ pruned to exclude all nodes $(a, b)$ with $b > m = \lceil \pi \sqrt{\frac{n}{3}} \rceil$. By Lemma 5.21 one can stop traversing a branch of $T_2$ or $T_3$ as soon as a node $(a, b)$ with $b > m = \lceil \pi \sqrt{\frac{n}{3}} \rceil$ is reached. Clearly, the time needed to generate these coprime pairs is $O(n)$.

Since every coprime pair must have components less than $m$, we know that

$$
\sum_{i=1}^{n} a_i \leq \sum_{i=1}^{n} \left\lceil \pi \sqrt{\frac{n}{3}} \right\rceil = O(n^{\frac{3}{2}}), \quad \text{and} \quad \sum_{i=1}^{n} b_i \leq \sum_{i=1}^{n} \left\lceil \pi \sqrt{\frac{n}{3}} \right\rceil = O(n^{\frac{3}{2}}).
$$

Therefore, a robot can build the ternary tree of coprime pairs, stopping a path as soon as it reaches a value exceeding $m$ in order to generate $n$ coprime pairs such that the perimeter complexity is $n(n^{\frac{3}{2}}) = O(n^{\frac{3}{2}})$. That is, a “good” set of coprime pairs can be generated efficiently.

Next, these coprime pairs have to be sorted in order of their slopes. This runs in
Thus, given \( n \) and \( i \), a robot can determine the coprime pair \((a_i, b_i)\) with the \( i \)th smallest slope in \( O(n \log n) \) time. We will refer to this algorithm for generating the \( n \) suitable coprime pairs as the Ternary Tree algorithm.

Recall that a robot can perform an arbitrary computation during the Compute part of its LCM cycle. To put this local computation of a robot in perspective, observe that each robot spends \( \Theta(n) \) time in its Look phase to generate the positions of all visible robots. In that backdrop, the \( O(n \log n) \) time is not excessive.

**Theorem 5.23** The Ternary Tree algorithm correctly solves OLCH with \( O(n^{3/2}) \) time and perimeter, \( O(n^3) \) area, and \( O(n \log n) \) local computation using 35 colors.

**Proof:** Starting from \( L_{OR} \), the Ternary Tree algorithm uses 6 colors: \texttt{moveUp}, \texttt{moveRight}, \texttt{moveLeft}, \texttt{nTop}, \texttt{nBottom}, and \texttt{done}. The other 29 colors were used in our previous line formation algorithm on a grid [14]. The upper bounds of time, perimeter, and area follow from Theorem 5.14.

---

**5.9.2. Improved Ternary Tree Algorithm**

We now detail an extension of the previous coprimes algorithm of Section 5.9, the Ternary Tree algorithm, such that the local computation needed to determine the \( i \)th smallest slope every LCM cycle is reduced to only linear time. The remaining algorithms differ from the previous coprimes algorithm only in their computation. Therefore, all perimeter, area, and time results immediately apply.

The model definition allows arbitrary computation in the “Compute” phase of each LCM cycle. However, constructing a ternary tree, pruning it, and sorting the values is
an expensive step requiring $O(n \log n)$ local time. Further, this must be performed every LCM cycle by every robot that has not terminated. Implementations would have to consider the complexity of the Compute phase of each LCM cycle.

Let the algorithm of this section used to determine the $i^{th}$ smallest slope in linear time be known as the Improved Ternary Tree algorithm.

**Lemma 5.24** For any two coprime pairs $(a, b)$ and $(c, d)$ such that $a < b < m$ and $c < d < m$, where $m = \left\lceil \pi \sqrt{\frac{\pi}{3}} \right\rceil$, $|\frac{b}{a} - \frac{d}{c}| \geq \frac{1}{m^2}$.

**Proof:** Consider any two coprime pairs $(a, b) \neq (c, d)$. With no loss of generality, let $\frac{b}{a} > \frac{d}{c}$. Then $bc > ad$ which implies $bc - ad \geq 1$ since $a, b, c, d \in \mathbb{Z}^+$. Clearly, $\frac{b}{a} - \frac{d}{c} = \frac{bc - ad}{ac} \geq \frac{1}{ac}$.

Both $a, c < m$ so $\frac{1}{ac} > \frac{1}{m^2}$, therefore $\frac{bc - ad}{ac} > \frac{1}{m^2}$. □

Given that no two coprime pairs can have slopes with a difference of less than $\frac{1}{m^2}$, a form of bucket sorting can be used to order the slopes where each bucket holds a range of slopes no larger than $\frac{1}{m^2}$.

**Lemma 5.25** Let $S = \{s_i : 0 \leq i < n\}$ be the set of slopes of the coprime pairs generated by the TT Algorithm. There exists an injective function $f : S \rightarrow \{0, 1, 2, \cdots, m^3\}$ such that $f(s_i) < f(s_j)$ iff $s_i < s_j$.

**Proof:** We start by observing that the minimum possible slope $s_{\text{min}} = \frac{m-1}{m-2}$ and the maximum slope $s_{\text{max}} = m - 1$. To ensure each bucket corresponds to a range of slopes no larger than $\frac{1}{m^2}$, we need $m^2 \left( (m - 1) - \frac{m-1}{m-2} \right)$ buckets that evenly divide the range of slopes. It is true that $m^2 \left( (m - 1) - \frac{m-1}{m-2} \right) < m^3 \forall m > 2$. If $m \leq 2$, there would not be more than one coprime pair, making it trivially sorted. With $m^3$ buckets holding slopes across the range $s_{\text{min}}$ to $s_{\text{max}}$, the coprime pairs can directly be mapped by slope in sorted order to a
bucket \( f(s_i) = \left\lfloor m^3 \left( \frac{s_i - s_{\min}}{s_{\max} - s_{\min}} \right) \right\rfloor \ \forall \ i \) upon generation. By Lemma 5.24, no two coprime pairs may be placed in the same bucket.

The proposed Improved Ternary Tree algorithm has the approach of taking the \( n \) coprime pair slopes \( s_i = \frac{y_i}{x_i} \) and converting them into \( n \) \( O(\log n) \)-bit integers whose order reflect the order of the slopes. By sorting these integers, we effectively sort the slopes.

Recall that for \( 1 \leq i \leq n \), the \( i \text{th} \) coprime pair is \((x_i, y_i)\) and its slope is \( s_i = \frac{y_i}{x_i} \). The slopes are lower and upper bounded by \( s_{\min} = \frac{m-1}{m-2} \) and \( s_{\max} = m - 1 \); that is, for all \( i, \ s_{\min} \leq s_i \leq s_{\max} \).

Given \( 1 \leq j \leq n \), the Improved Ternary Tree algorithm has the following steps.

1. Using the function \( f(s_i) = \left\lfloor m^3 \left( \frac{s_i - s_{\min}}{s_{\max} - s_{\min}} \right) \right\rfloor \) discussed in Lemma 5.25, create a triplet \((f(s_i), x_i, y_i)\) for each coprime pair \((x_i, y_i)\).

2. Sort the triplets \((f(s_i), x_i, y_i)\) in ascending order of the value of \( f(s_i) \).

3. If \( f(s_i) = j \), then the coprime pair with the \( j \text{th} \) smallest slope is \((x_i, y_i)\).

Step 1 requires \( O(n) \) time. By Lemma 5.25, the order of values of \( f(s_i) \) reflect the orders of the slopes of the coprime pairs; further, no two coprime pairs \( i, i' \) have the same \( f(s_i), f(s_i') \). This implies the correctness of Step 3. The sorting required for Step 2 needs \( O(n) \) time as the \( f(s_i) \)s are \( \log m^3 = O(\log n) \) bits long.

Lemma 5.26 Given \( n \) and \( i \), a robot can determine the coprime pair with the \( i \text{th} \) smallest slope that was generated using the ITT algorithm in \( O(n) \) time.

Thus, we have reduced the computation needed for each robot to determine the \( i \text{th} \) smallest slope every epoch from \( O(n \log n) \) to \( O(n) \). The number of colors remains the same as in the Ternary Tree algorithm.

Theorem 5.27 The Improved Ternary Tree algorithm correctly solves OLCH with \( O(n_3^2) \)
time and perimeter, $O(n^3)$ area, and $O(n)$ local computation using 35 colors in asynch. □

5.10. Farey Sequence Algorithm

Having achieved linear time computation to determine the $i^{th}$ smallest slope, we can still reduce the computation further using an alternative method of generating and selecting slopes. We will refer to the algorithm of this section as the Farey Sequence algorithm. Recall that the goal of the ternary tree method was to generate $n$ unique coprime pairs $(a_i, b_i)$ for $1 \leq i \leq n$ such that $a_i < b_i$ and to order these coprime pairs by increasing slope $\frac{b_i}{a_i}$. We now outline an approach to reduce the local computation time of selecting the $i^{th}$ smallest slope given $i$ and $n$ to sublinear time [75].

The Farey sequence of order $m$, denoted $F_m$, is defined as the sequence of completely reduced fractions $c_i/d_i$ between 0 and 1 (ensuring $c_i < d_i$) in increasing order with denominators less than or equal to $m$.

This sequence exhausts all coprime numbers such that $c_i < d_i < m$ without duplicates. The elements in the sequence are already sorted in increasing order. The same argument used for the ternary tree method based on Euler's totient function can be made to show that $n$ coprime numbers will be generated. The problem remains to select the the $i^{th}$ smallest slope, or the $i^{th}$ element in the sequence. This problem is referred to as the order statistics problem for Farey sequences. The order statistics problem is defined as follows: Given $m$ and $k$, select the $k^{th}$ element from $F_m$. This is exactly what each robot must do to determine its goal location during an LCM cycle. This order statistics problem can be solved in $O(m^{3/4} \log m)$ computation time for a Farey sequence of order $m$, as shown by Pawlewicz [76]. The algorithm of Pawlewicz relies on binary search and uses the rank
problem to determine if the target is higher or lower for each step of the search. The rank problem is defined as finding the number of fractions in $F_n$ that are less than or equal to $x$ given a positive integer $n$ and a real number $x \in [0,1]$. Given that $m$ is approximately $\sqrt{n}$, the $i^{th}$ smallest slope of $n$ suitable coprime pairs can be computed in $O(n^{\frac{3}{8}} \log n)$ computation time, a significant improvement on linear computation. However, to determine the desired value of $m$, robots must first determine $n$ which takes $O(n)$ time. Therefore, although this further reduces the local computation needed to select the $i^{th}$ smallest slope, it does not reduce the asymptotic complexity of the overall algorithm unless parallel processing is used. Therefore, this can be a valuable alternative in the case of robots equipped with multi-core processors or in cases where the fastest possible local computation is desired over ease of implementation. The number of colors remains the same as in the Ternary Tree and Improved Ternary Tree algorithms.

**Lemma 5.28** The Farey Sequence algorithm uses $O\left( n^{\frac{3}{8}} \log n \right)$ local computation to select the $i^{th}$ smallest slope given $i$ and $n$.

**Proof:** The order statistics problem can be solved in $O(m^{\frac{3}{4}} \log m)$ time for a Farey sequence of order $m$, as shown by Pawlewicz [76]. As in the Ternary Tree algorithm, we let $m = \lceil \pi \sqrt{\frac{n}{3}} \rceil = O(\sqrt{n})$. Thus, $O(n^{\frac{3}{8}} \log n)$ computation time is used. □

**Theorem 5.29** The Farey Sequence algorithm correctly solves OLCH with $O(n^{\frac{3}{2}})$ time and perimeter, $O(n^{3})$ area, and $O\left( n^{\frac{3}{8}} \log n \right)$ local computation using 35 colors. □

5.11. Conclusion

In this chapter, we have presented three algorithms (namely, Quadratic, Ternary Tree, and Farey Sequence) for placing robots on the corners of a convex hull on a grid
(with no robots inside the convex hull). We have developed lower bounds to establish that two of our algorithms (not including the original ternary tree method) are optimal in time, distance, perimeter, and area complexity. A summary of these Convex Hull Formation algorithms is provided in Table 5.1.

As noted in Chapter 2, the unit distance in the grid reflects both the resolution of distance and speed (distance per unit time) of robots. Such an assumption is well justified as distance and speed are clearly related to each other. However, in some cases, one may need a higher resolution for speed than for the distance (for example, with relatively large robots, with little variations in speed). This is simple to handle, however, if the unit of distance $d$ and that for speed $s$ are related by a constant $c = \frac{d}{s}$; simply expand the set of colors by a factor of $c$ and break each movement to a sequence of $c$ movements. Furthermore, it would be interesting to reduce the number of colors used in our algorithms.
Chapter 6. Doorway Egress by Point Robots

In this chapter, we introduce the problem of Doorway Egress. In this problem, the 2-dimensional real plane is bisected by an infinite line or wall. The wall contains a single gap or doorway. Let the robots begin in any arbitrary but unique positions on one side of the wall. The goal is to move the robots through the doorway as quickly as possible. We investigate this problem across many combinations of the model variants previously discussed. Specifically, we vary the visibility, extent, lights, and synchronization (discussed in Section 6.4) of the point robots model. We then provide several algorithms with various optimalities and a variety of model variants. In the following chapter, we similarly discuss Doorway Egress for fat robots.

Doorway Egress is fundamental as it simulates the evacuation of robots that have completed their task or the infiltration of robots into a target space. For this reason, we provide algorithms for Doorway Egress that can be appended to other robot swarm algorithms on many common models.

6.1. Contributions of This Chapter

In this chapter, we introduce the problem of Doorway Egress. We then establish four results (summarized in Table 6.1).

Before we proceed to the results, we introduce some notation. Recall that a robot swarm can have point robots or fat robots (see Chapter 2). Independently, the swarm can operate according to different activation schedules. In this chapter, we only consider either ASYNC or SSYNC. Further independently, the swarm can have robots that are transparent (affording complete visibility in all configurations) or opaque. Finally, recall that
Table 6.1. Summary of DOORWAY EGRESS Algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Extent</th>
<th>Colors</th>
<th>Visibility</th>
<th>Schedule</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Point</td>
<td>−</td>
<td>Any</td>
<td>Any</td>
<td>Ω(1)</td>
</tr>
<tr>
<td>{V, S, L}</td>
<td>Point</td>
<td>−</td>
<td>Obstructed</td>
<td>\ASYNC</td>
<td>(O(n))</td>
</tr>
<tr>
<td>{\nabla V, S, L}</td>
<td>Point</td>
<td>10</td>
<td>Global</td>
<td>\ASYNC</td>
<td>(O(1))</td>
</tr>
<tr>
<td>{V, S, L}</td>
<td>Point</td>
<td>−</td>
<td>Obstructed</td>
<td>SSYNC</td>
<td>(O(1))</td>
</tr>
</tbody>
</table>

robots can optionally have lights.

These “abilities” (visibility, synchrony, lights), abbreviated in an “ability set” as \{V, S, L\}, provide the framework in which an algorithm operates. Each element of the ability set is a Boolean variable asserted (or not) by the symbols \(V\), \(S\), and \(L\) (or \(\nabla V\), \(\nabla S\), and \(\nabla L\)). Therefore, for example, the set \{V, S, L\} indicates a swarm with “visibility” (i.e., transparency) but no lights and operating in \ASYNC. Recall that, in addition to the ability set, robots can vary by extent (point or fat).

**Result 1:** A \(O(n)\) time algorithm for point robots that are opaque and operate in \ASYNC (without lights). Therefore, this algorithm runs correctly in \(O(n)\) time for any of the model variants we discuss.

**Results 2-4:** Three \(O(1)\) time, and therefore, optimal, algorithms for point robots with each algorithm making use of exactly one of \{visibility, synchrony, lights\}.

All algorithms are shown to be collision-free.

### 6.2. Previous Work

The most closely related previous work is the problem of Evacuation [77, 78, 79, 80]. Evacuation is defined as follows: \(k\) robots must evacuate, or exit, from a unit circle by finding an unknown exit point on the perimeter of the circle and moving to it. In contrast to DOORWAY EGRESS, the robots in Evacuation are allowed to communicate with other robots arbitrarily, e.g., some robot may inform all other robots of the position of the door.
when it finds it. Furthermore, exit point is not known in Evacuation whereas the doorway is known to the robots if not obstructed. Evacuation has been well-studied in the literature in the presence of faulty robots and in many different domains, such as triangles, lines, etc.

We have described the problem of Filling in Chapter 3. Filling can be seen as direct opposite of Doorway Egress except the width of the doorway; Filling asks the robots to enter one by one from the door, whereas Doorway Egress asks the robots to exit (possibly in parallel) from the doorway.

6.3. Chapter Roadmap

We discuss the model variants and preliminaries in Section 6.4. Result 1 (Algorithm $\{V, S, L\}$) is described in Section 6.5. Sections 6.6, 6.7, 6.8 describe Results 2-4 (Algorithms $\{V, S, L\}$, $\{V, S, L\}$, and $\{V, S, L\}$, respectively, while Section 6.9 handles a special case for the algorithms of Results 2-4. Section 6.10 then provides a brief discussion.

6.4. Doorway Egress

In this section, we formally define Doorway Egress and provide the details of the model variants used in this chapter.

The “base model” that we now describe refers to the details of the model that are common across all model variants, i.e., any combination of abilities. As before, we consider a system of $n$ robots $Q = \{r_0, r_1, \cdots, r_{n-1}\}$. Each robot is a dimensionless point robot that can move in an infinite 2-dimensional real plane $\mathbb{R}^2$. The real plane is, however, divided into by an infinite, straight line or wall, labelled $W$, that acts as an obstruction to the robots. There is a single gap in the wall, a doorway between sides of the wall, in the
form of a line segment of length $2s$. Recall that the robots have no access to a common coordinate system and do not know the value of $n$. Further, a robot $r_i$ can see another robot $r_j$, and vice-versa, iff there is no third robot $r_k$ on the line segment joining $r_i$ and $r_j$. Robots are initially placed arbitrarily, but in unique positions, on one side of the wall.

The **Doorway Egress** Problem models the movement of robots on one side of a wall to the other side through the doorway. Each robot first positions itself on the line segment representing the doorway before crossing over to the other side immediately or at a subsequent epoch. When robots cross over to the other side of the wall, they “disappear” spontaneously. A solution to **Doorway Egress** is to cause all robots to disappear across the wall. Robots’ movements are restricted to straight paths by design. The robots may be capable of moving along curved paths, but the algorithms would not make use of this ability.

**Doorway Egress** has applications in guiding a swarm of robots through a doorway. It can be generalized to multiple doorways, with wall segments as obstacles placed in a line.

As previously stated, the real plane is bisected by a wall $W$ that contains a doorway. We can split the plane into two half-planes, $H_a$ and $H_b$, by an infinite line through $c$ that is perpendicular to the wall.

For the point robots model, we describe the configuration of robots as follows. Consider all lines from the center of the door, $c$, through each robot. For the two half-planes, we will have lines $\ell_{a,1} \cdots \ell_{a,j}$ and $\ell_{b,1} \cdots \ell_{b,k}$ for $0 \leq j, k \leq n$. On each half-plane, the lines are ordered based on increasing $\angle(W_a, c, \ell_{a,i})$ or $\angle(W_b, c, \ell_{b,i})$ where $W_a$ is the side of the wall on $H_a$ and $c$ is the center of the doorway. The same notation can be used with
respect to the center points of fat robots. The notation used is illustrated in Figure 6.1.

![Figure 6.1. Basic Terminology of Robot Configurations](image)

Given the symmetry of the two half-planes, we will often only consider only one half-plane, and the algorithms will operate independently on each half-plane. In Section 6.9, we will discuss the special case of robots initially lying on the half-plane line. Since the algorithm is symmetric for each half-plane, and each half-plane operates independently, we will simply use $\ell_i$ instead of $\ell_{a,i}$ or $\ell_{b,i}$ without loss of generality.

On each half-plane, the orientation is as follows: up and down are parallel to the wall, with up being towards the center; left and right are perpendicular to the wall, with right being towards the wall. The robots are not inherently oriented and must determine these directions; however, this terminology will make the movement of the algorithms clearer.

### 6.5. Doorway Egress Without Additional Capabilities

Consider the case of robots with no abilities, i.e. $\{V, S, L\}$. In this section, we provide a linear-time, collision-free algorithm for both point as described in Section 6.4 with
no abilities.

A trivial algorithm, called Algorithm $\{V, S, L\}$, running in $O(n)$ time would move
the innermost robot (closest to $c$) with the smallest angle $\angle(W, c, r_i)$, call it a leader,
through the doorway along the line $\overline{cr_i}$. As the leader moves towards the doorway, it
retains its status as leader until it disappears since its distance to $c$ strictly decreases as it
moves. This algorithm moves robots sequentially through the doorway, at least one robot
each epoch. Therefore, the algorithm requires $O(n)$ time. Furthermore, since each robot
ultimately moves in a straight line from its initial position to the center of the door, the
distance complexity is $O(D)$, where $D$ is the diameter of the initial configuration. Note
that both the doorway and the robots are considered when determining the diameter of a
configuration.

**Theorem 6.1** Algorithm $\{V, S, L\}$ solves Doorway Egress in ASYNC in $O(n)$ time
with $O(D)$ distance complexity for point robots. □

6.6. Opaque Robots With Lights in ASYNC

Recall that the initial configuration consists of point robots in $\mathbb{R}^2$ on one side of an
infinite wall having a single door of width $2s$. Because robots are points, no robot can ob-
struct others from viewing the position of the wall or even the entire doorway. Therefore,
all robots can determine the center of the doorway, $c$, as well. Thus, point robots, even
with no capabilities, can determine the directions up, down, left, and right at any time for
themselves.
6.6.1. Phase 1 - Position Beacons

The goal of the first phase is to place special guiding robots called *beacons*. Beacons will be used in the following phase to lead their corresponding non-beacons to unique lines from the center of the doorway, $c$.

With all robots initially colored Initial, there are two cases for a robot to become a beacon:

1. A robot $r_i$ is on line $\ell_i$ with at least one other robot, $\ell_{i+1}$ has multiple robots or does not exist, and $r_i$ is furthest from $c$ on $\ell_i$.

2. A robot $r_i$ is on line $\ell_i$ with no other robot and $\ell_{i-1}$ has multiple robots.

The two cases account for either possibility regarding a line containing multiple robots, i.e., either the subsequent line $\ell_{i+1}$ contains exactly one robot, or it does not. That is, the next line may have multiple robots or there may be no subsequent line containing robots. If there is only one, then that’s the beacon. Otherwise, move a single robot out between $\ell_i$ and $\ell_{i+1}$ and then that robot falls into the previous case. In either situation, the beacon robot colors itself Moving1 and moves to the closest point satisfying the following conditions:

- It is on a line $\ell_k$ such that $\angle(\ell_{k-1}, c, \ell_k) = \angle(\ell_k, c, \ell_{k+1})$. That is, its line splits the angle between adjacent lines.

- It is farther from $c$ than robots on adjacent lines.

- After corresponding non-beacons move towards it in Phase 2, they will be strictly farther than $\frac{s}{3}$ from $c$.

The third condition will be elucidated in Sections 6.6.2 and 6.6.3. If line $\ell_i$ is the closest line to the half-plane line, then treat $\ell_{i+1}$ as the half-plane line. Since the beacon is strictly moving in the empty regions between the adjacent lines of its initial configuration, the beacon moves into its position in one LCM cycle once activated. An illustration of Phase 1 is provided in Figure 6.2.
The goal of this phase is to move the robots on a line containing multiple robots towards the beacon to unique lines from $c$.

Before the non-beacons are guided to unique lines by their respective beacon, the innermost robot on a line with multiple robots changes its color to Lock. Since all of the other robots on the line will move towards the beacon, the robot colored Lock need not move in this phase.

Once the beacon robot colored Moving1 sees the previous line has a robot colored Lock, it changes its color to Beacon1.

With $\ell_{i+1}$ containing only a robot colored Beacon1, the robots on $\ell_i$ that aren’t the innermost robot (lock) change their color to Moving2 and move directly towards the beacon. This ensures that no non-beacon robot in motion can obstruct another robot’s view of the beacon; otherwise would imply two robots shared the same location on $\ell_i$.

Each non-beacon robot $r_i$ moves directly towards the beacon from $\ell_i$ to a new line.
$\ell_i$ towards the beacon’s line $\ell_b$ such that $\angle(\ell_i, c, \ell_i')$ increases as $r_i$’s initial distance from $c$ decreases. Let $d(r_i, c)$ denote the euclidean distance between robot $r_i$ and the center of the door $c$. Let $\theta' = \angle(\ell_i, c, \ell_i')$ and $\theta = \angle(\ell_i, c, \ell_b)$. Then, specifically, each robot moves such that

$$\theta' = \left(1 - \frac{d(r_i, c)}{d(r_b, c)}\right) \theta$$

Since a beacon robot moves in Phase 1 such that $d(r_i, c) < d(r_b, c)$, each non-beacon robot moves to a new line strictly between $\ell_i$ and $\ell_b$.

A robot $r_i$ colored $\text{Moving2}$ keeps track of it’s corresponding beacon robot $r_b$, and thus $\ell_b$, throughout this portion of Phase 2 since the non-beacons move to avoid obstructing others from seeing the beacon and only robots colored $\text{Moving2}$ or $\text{Lone1}$ lie between $\ell_i$ and $\ell_b$. The described movement is illustrated in Figure 6.3.

![Figure 6.3. Movement of Non-Beacons Towards the Beacon](image)

When a robot wakes up colored $\text{Moving2}$, it changes its color to $\text{Lone1}$ no matter what. This is because it would have already moved to its unique line from $c$ in its previous
LCM cycle. Therefore, the color indicates to the beacon that it has completed its movement to a unique line.

Once a robot colored Lock is on a line by itself, i.e. all non-beacons have moved off its line towards the corresponding beacon, it moves along its line to a point where it has the same distance from $c$ as the beacon. Its visibility of the beacon is not obstructed for the same reason no two non-beacons obstruct the other’s view of the beacon.

The beacon colored Beacon1 then waits until all of its non-beacons (all robots between its line and and the line of its locking robot, the closest robot colored Lock on line $\ell_l$ such that $l < b$) are colored Lone1. Once that is satisfied, and the locking robot is in position such that $d(r_l, c) = d(r_b, c)$, the beacon then changes its light to Beacon2 to indicate to all of its non-beacons that they have reached their unique lines and are now free to continue to the next portion of the algorithm.

![Figure 6.4. The Beacon Informs Non-Beacons They Have Reached Unique Lines](image-url)
6.6.3. Phase 3 - Move Robots to Unique Heights In Front of Door

When a robot colored \texttt{Lone1} sees its beacon is colored \texttt{Beacon2}, it changes it’s color to \texttt{Lone2}, indicating that it knows all non-beacons corresponding to its beacon are on their unique lines.

Once a robot is colored \texttt{Lone2}, the beacon is no longer relevant to it. It moves along its current line $\ell_i$ until $d(r_i, c) = \frac{s}{3}$.

A robot $r_i$ on line $\ell_i$ colored \texttt{Lone1} keeps track of its corresponding beacon robot $r_b$, and thus $\ell_b$, since only robots colored \texttt{Lone1} or \texttt{Lone2} lie between $\ell_i$ and $\ell_b$. No moving robot colored \texttt{Lone2} blocks a robot colored \texttt{Lone1} from seeing it’s beacon because the third condition for a beacon’s position in Phase 1 is such that non-beacons will move towards it and be strictly farther than $\frac{4}{3}$ from $c$. In the subsequent phase, when robots move to exactly $d(r_i, c) = \frac{s}{3}$, this guarantees that all robots colored \texttt{Lone2} move towards $c$ along their current lines and the mapping of non-beacon robots to their new lines ensures that a robot moving towards $c$ along its line will not cross a line segment between any robot colored \texttt{Lone1} and its beacon.
The beacon colored \texttt{Beacon2} on line $\ell_b$ waits until all visible robots between it and its locking robot are colored \texttt{Lone2} and at distance $\frac{s}{3}$ from $c$. Note that if a robot is not visible to the beacon, then it must have already changed its color to \texttt{Lone2} and moved towards $c$ because it was originally guaranteed to be visible to the beacon when colored \texttt{Lone1}, and even if a moving robot colored \texttt{Lone2} were blocking a robot colored \texttt{Lone1} that had yet to move, the blocking robot $r_i$ must be in a position such that $d(r_i, c) > \frac{s}{3}$. Once the conditions are satisfied, the beacon changes its color to \texttt{Beacon3}.

The locking robot sees a robot on a subsequent line with the same distance from $c$ colored \texttt{Beacon3}, with only robots colored \texttt{Lone2} at distance $\frac{s}{3}$ from $c$ in between. It then moves along its current line to $\frac{s}{3}$ from $c$. 
The beacon colored Beacon3 on \( \ell_i \) looks. Recall the beacon moves halfway between adjacent lines in Phase 1. Therefore \( \angle(\ell_i, c, \ell_{i+1}) = \angle(\ell_l, c, \ell_i) \), where \( \ell_l \) is the line containing the associated locking robot. The beacon can use this to determine the line that should contain the locking robot whether or not it is visible. If there is no robot on \( \ell_l \) with the same distance from \( c \) colored Lock, then the locking robot must have already begun moving to \( \frac{s}{3} \) from \( c \), and then so does the beacon.

6.6.4. Phase 4 - Move Robots Through the Door

Once the non-beacons, locking robot, and beacon have moved to \( \frac{s}{3} \) from \( c \), they are on the circumference of one quadrant of a circle, as shown in Figure 6.7. This ensures that they are at unique heights, able to move directly rightwards and reach the door, and can see all other robots on the arc (if there is no robot \( r_i \) such that \( d(r_i, c) < \frac{s}{3} \)).

When a robot and robots on neighboring lines (or just neighbors) meet the following criteria:

- On a unique line
- Colored Beacon3, Lone2, or Lock
• At a distance $\frac{s}{3}$ from $c$

It changes its color to **Final**.

When a robot only sees robots colored **Final**, it moves horizontally through the door.

![Figure 6.7. Robots Move Horizontally Through the Door](image)

The robots can disappear as soon as they wish. It can be immediately upon waking up in the doorway or once only robots in the doorway are visible, i.e. all robots have reached the door. For the sake of uniformity, the algorithm is such that once all *visible* robots are in the doorway, which implies all robots are in the doorway, the robots disappear and terminate.

6.6.5. **Algorithm** $\{V, S, L\}$

The algorithm is provided in the form of condition-action pairs. They are separated by color purely for ease of comprehension.

**Robots colored Initial:**
**Condition I1:** Robot $r_i$ is colored Initial, has multiple robots on its current line, $r_i$ is furthest from $c$ on its current line, and either the next line has multiple robots or there is no subsequent line containing robots.

**Action I1:** Change color to Moving1 and move to the closest point on a new line halfway between adjacent lines $\ell_i$ and $\ell_{i+1}$ such that $r_i$ is farther from $c$ than robots on adjacent lines and non-beacons will be farther than $\frac{4}{3}$ from $c$ after moving towards the beacon. I.e. move to beacon position.

**Condition I2:** Robot $r_i$ is colored Initial, is on a unique line from $c$, and there are multiple robots on the previous line.

**Action I2:** Change color to Moving1 and move to the closest point on a new line halfway between adjacent lines $\ell_i$ and $\ell_{i+1}$ such that $r_i$ is farther from $c$ than robots on adjacent lines and non-beacons will be farther than $\frac{4}{3}$ from $c$ after moving towards the beacon. I.e. move to beacon position.

**Condition I3:** Robot $r_i$ is colored Initial, has multiple robots on its current line, is not the closest robot to $c$ on the line, and there is a robot colored Beacon1 on a subsequent line with only robots colored Moving2 or Lone1, if any, in between $r_i$’s line and the beacon’s.

**Action I3:** Change color to Moving2 and move directly towards the beacon to a unique line creating the proper angle from the current line, i.e. $\theta' = \left(1 - \frac{d(r_i,c)}{d(r_b,c)}\right) \theta$.

**Condition I4:** Robot $r_i$ is colored Initial, is on a line with multiple robots, and is closest to $c$ on its line.
Action I4: Change color to Lock

Condition I5: Robot \( r_i \) is colored Initial, is on a line by itself, and the previous line contains only one robot.

Action I5: Change color to Lone2

Robots colored Moving1:

Condition M1: Robot \( r_i \) is colored Moving1 and sees a robot colored Lock on the previous line

Action M1: Change color to Beacon1

Robots colored Moving2:

Condition N1: Robot \( r_i \) is colored Moving2

Action N1: Change color to Lone1

Robots colored Beacon1:

Condition B1: Robot \( r_i \) on line \( \ell_b \) is colored Beacon1, all robots between \( \ell_b \) and the \( \ell_t \), the line of the closest robot colored Lock such that \( l < b \), are colored Lone1, and the aforementioned locking robot \( r_l \) satisfies \( d(r_l, c) = d(r_i, c) \).

Action B1: Change color to Beacon2

Robots colored Beacon2:

Condition C1: Robot \( r_i \) is colored Beacon2, all robots between \( \ell_b \) and the \( \ell_t \), the line of the closest robot colored Lock such that \( l < b \), are colored Lone2 and at distance \( \frac{2}{3} \) from \( c \), and the aforementioned locking robot \( r_l \) satisfies \( d(r_l, c) = d(r_i, c) \).

Action C1: Change color to Beacon3
Robots colored Beacon3:

Condition D1: Robot \( r_i \) on line \( \ell_i \) is colored Beacon3, there is not a robot on line \( \ell_l \) colored Lock with the same distance from \( c \) as \( r_i \). (Recall \( r_i \) can determine \( \ell_l \) because \( \angle(\ell_i, c, \ell_{i+1}) = \angle(\ell_l, c, \ell_i) \))

Action D1: Move along current line to distance \( \frac{s}{3} \) from \( c \)

Condition D2: Robot \( r_i \) is colored Beacon3, it and it’s neighbors are on lines by themselves at distance \( \frac{s}{3} \) from \( c \).

Action D2: Change color to Final

Robots colored Lone1:

Condition L1: Robot \( r_i \) is colored Lone1, and the closest subsequent line not containing only robots colored Lone1 or Lone2 contains a robot colored Beacon2

Action L1: Change color to Lone2

Robots colored Lone2:

Condition K1: Robot \( r_i \) is colored Lone2 and not at distance \( \frac{s}{3} \) from \( c \)

Action K1: Move along current line to distance \( \frac{s}{3} \) from \( c \)

Condition K2: Robot \( r_i \) is colored Lone2, it and it’s neighbors are on lines by themselves at distance \( \frac{s}{3} \) from \( c \).

Action K2: Change color to Final

Robots colored Lock:

Condition O1: Robot \( r_i \) is colored Lock, is on a line by itself, and sees a robot on a subse-
quent line colored \textit{Beacon1} with only robots colored \textit{Moving2} or \textit{Lone1}, if any, in between $r_i$’s line and the beacon’s.

**Action O1**: Move along current line to a distance from $c$ equal to the beacon’s

**Condition O2**: Robot $r_i$ is colored \textit{Lock}, $d(r_i, c) > \frac{s}{3}$, and $r_i$ sees a robot on a subsequent line colored \textit{Beacon3} with the same distance from $c$ and only robots colored \textit{Lone2} at a distance $\frac{s}{3}$, if any, in between $r_i$’s line and the beacon’s.

**Action O2**: Move along current line to distance $\frac{s}{3}$ from $c$

**Condition O3**: Robot $r_i$ is colored \textit{Lock}, it and it's neighbors are on lines by themselves at distance $\frac{s}{3}$ from $c$.

**Action O3**: Change color to \textit{Final}

**Robots colored Final**:

**Condition F1**: Robot $r_i$ is colored \textit{Final}, and only robots colored \textit{Final} are visible

**Action F1**: Move rightwards through the doorway

**Condition F2**: All visible robots are in the doorway

**Action F2**: Disappear and terminate

This algorithm uses 10 colors, which can be observed in the condition-action pairs above. We subsequently analyse the other performance metrics for Algorithm $\{V, S, L\}$.

6.6.6. Analysis

**Lemma 6.2** Robots colored initial can determine whether they are a beacon, and non-beacons colored initial can determine their corresponding beacon.

**Proof**: There are two cases for a robot to become a beacon:

89
1. A robot \( r_i \) is on line \( \ell_i \) with at least one other robot, \( \ell_{i+1} \) has multiple robots or does not exist, and \( r_i \) is furthest from \( c \) on \( \ell_i \).

2. A robot \( r_i \) is on line \( \ell_i \) with no other robot and \( \ell_{i-1} \) has multiple robots.

The criteria for determining a beacon for each line \( \ell_i \) having multiple robots only depends on knowledge of: adjacent lines, if a robot is the farthest robot from \( c \) on \( \ell_i \), and if there are multiple robots on \( \ell_i \). A robot can see every robot on its adjacent lines, since there can be no robots on a line between them to obstruct their visibility. All robots know the location of \( c \), so robots know their line and thus can see if they are on it by themselves and if they are the outermost robot on their line. Thus, a robot can determine whether it is a beacon.

As the beacon moves into position, all robots on line \( \ell_i \) see a robot colored Moving1 on the next line \( \ell_{i+1} \) and do nothing until it changes color. Immediately after the beacon is colored Beacon1, the non-beacons can see it on \( \ell_{i+1} \). They change their color to Moving2 and move into the region between \( \ell_i \) and \( \ell_{i+1} \). Since they move directly towards the beacon, a robot colored Moving2 cannot obstruct the view of a robot still on \( \ell_i \). Otherwise would imply that they shared the same position on \( \ell_i \), but we assumed distinct starting positions. Therefore, a robot still colored Initial on \( \ell_i \) can see its beacon once it is in place.

A robot colored Moving2 may change its color to Lone1 within the region between \( \ell_i \) and the beacons line, but a robot colored Lone1 does not change its color to until all non-beacons corresponding to the same beacon have moved. Suppose, for the purpose of contradiction, that a robot \( r_i \) on \( \ell_i \) colored Initial moved towards the wrong robot \( r_b \) on \( \ell_b \) colored Beacon1. Then there are only robots colored Moving2 or Lone1 between \( \ell_i \) and
\ell_b$. However, there would be some innermost robot on a line $\ell_j$ whose true beacon is $r_b$ such that $\ell_j$ is closer to $\ell_b$. This robot may be colored \texttt{Initial} or \texttt{Lock}, depending on its phase, but then $r_i$ would not have chosen $r_b$ as a beacon; a contradiction. If the aforementioned robot has proceeded past being colored \texttt{Initial} or \texttt{Lock} to being colored \texttt{Final}, then the corresponding beacon would no longer be colored \texttt{Beacon1} due to Condition O2. Thus, a non-beacon colored \texttt{Initial} will not mistake its corresponding beacon, whether or not its true beacon is in place. 

\textbf{Lemma 6.3} \textit{Beacons move into position (Phase 1) in $O(1)$ epochs in Algorithm \{\texttt{V}, \texttt{S}, \texttt{L}\}.}

\textbf{Proof:} Recall the two cases for a robot to be designated a beacon stated in Lemma 6.2. A beacon robot $r_b$ on line $\ell_i$ in Case 1 changes its color to \texttt{Moving1} and moves into the empty space between $\ell_i$ and $\ell_{i+1}$ without waiting in 1 epoch. A beacon robot $r_b$ on line $\ell_i$ in Case 2 changes its color to \texttt{Moving1} and moves strictly between $\ell_{i-1}$ and $\ell_{i+1}$ without waiting in 1 epoch. That is to say, the beacon is in place in 1 epoch in either case.

Meanwhile, a robot colored \texttt{Initial} sees it is the innermost robot on it’s line containing multiple robots, and changes its color to \texttt{Lock} in 1 epoch. Subsequently, the beacon colored \texttt{Moving1} satisfies Condition M1 and changes its color to \texttt{Beacon1} in 1 epoch. This completes Phase 1 for the beacon and its corresponding lock and non-beacons. 

\textbf{Lemma 6.4} \textit{No robot in Phase 1 collides with another robot in Phase 1.}

\textbf{Proof:} The beacons in Case 1 move within empty, non-overlapping regions. That is, there is one beacon robot for line $\ell_i$, which moves strictly between $\ell_i$ and $\ell_{i+1}$. A collision between beacons would imply that one line has two beacons, resulting in a contradiction.
since there cannot be two outermost robots on $\ell_i$. The beacons in Case 2 move strictly between adjacent lines. By the same logic as that of Case 1, no other beacons enter this region. Only beacon robots move in Phase 1.

\[\square\]

**Lemma 6.5** Once a beacon is colored Beacon1, completing Phase 1, corresponding non-beacons move to form unique lines from $c$ and the lock moves to position (Phase 2) in $O(1)$ epochs in Algorithm $\{V, S, L\}$.

**Proof:** The non-beacons initially see their beacon on the next line colored Beacon1, indicating that they are free to move towards it.

Robots on $\ell_i$ change their color to Moving2 and move directly towards the beacon. Since these robots all move along non-overlapping lines towards the beacon robot, no robot can have its view of the beacon obstructed by a robot moving towards it. Further, even as non-beacons move and change their color, the beacon is known to all of its non-beacons as shown in the proof of Lemma 6.2. To get each robot to a unique line from $c$, non-beacons map their distance from $c$ to a percentage of the angle between $\ell_i$ and $\ell_b$, the line containing the beacon. Since the beacon is guaranteed to be further from the center than any moving non-beacon, the Euclidean distance $d(r_b, c) > d(r_i, c)$ for any robot $r_i$ on $\ell_i$. Further, given that $d(r_i, c)$ is unique for all $r_i$ on $\ell_i$, they all move to unique line between $\ell_i$ and $\ell_b$. Therefore, each non-beacon determines a unique angle and moves to a unique line from $c$ in 1 epoch once the non-beacons are free to move.

Once a robot wakes up colored Moving2, it changes its color to Lone1 in 1 epoch since it has already moved to a unique line with one movement. No other robot sharing the same beacon will move to the same line since that would imply the same distance from $c$ along $\ell_i$, i.e., it would imply the same initial position.
Once the robot $r_l$ colored Lock is left on $\ell_i$ by itself, it moves out along its current line until $d(r_l, c) = d(r_b, c)$ in 1 epoch.

The beacon colored Beacon1 waits until all robots between it and the closest robot on a prior line colored Lock are colored Lone1 and the lock satisfies $d(r_l, c) = d(r_b, c)$. It then changes its light to Beacon2 in 1 epoch.

The lock was initially not obstructed as previously shown, and at the end of the movement it is non obstructed since $d(r_l, c) = d(r_b, c)$ and all robots between satisfy $d(r_b, c) > d(r_i, c)$ as previously shown. Suppose that the lock is in motion and the beacon’s view of it is obstructed. Then its true non-beacons are still visible to it, and there is a strictly larger set of robots between it and the next closest lock (on a prior line) for it to wait for. Thus, if the beacon still sees only robots colored Lone1 between it and the visible lock, then its true non-beacons are certainly in place. Thus, the beacon will not change its color too early by looking while its true lock is in motion.

\begin{lemma}
No robot in Phase 2 collides with another robot in Phases 1 or 2.
\end{lemma}

\begin{proof}
The non-beacon robots move along unique, non-overlapping paths towards the beacon. Thus, non-beacons do not collide with each other.

The beacon does not move in this phase. Thus, it does not collide with any robot, unless another robot moves to its position.

Recall that each non-beacon robot moves such that $\theta' = \left(1 - \frac{d(r, c)}{d(r_b, c)}\right) \theta$. Since a non-beacon $r_j$ satisfies $d(r_j, c) < d(r_b, c)$ due to the movement of $r_b$ in Phase 1, each non-beacon robot $r_i$ moves to a new line strictly between $\ell_i$ and $\ell_b$. Thus, the non-beacons do not collide with the beacon.

No collisions occur with robots in previous phases since only beacons move in
Phase 1, and if a beacon were in the region between $\ell_i$ and $\ell_b$, that would contradict the assumption that $r_b$ is the beacon for robots on $\ell_i$. The same logic can be used to explain why there are no non-beacons that corresponding to a different beacon that collide with the $r_b$ or its non-beacons. That is, $r_b$ and its corresponding non-beacons and lock will not collide with robots outside of this set of robots.

The robot colored Lock does not collide with any robot as it moves out along $\ell_i$ because it waits until all other robots have moved off this line. The non-beacons cannot travel back towards $\ell_i$ once they have moved. No robot in Phases 1 or 2 moves to reach a line that already contains a robot, as previously shown. Thus, no robot collides with the robot colored Lock in this phase.

\[ \square \]

\textbf{Lemma 6.7} Once a beacon and its corresponding non-beacon robots and lock are on unique lines from $c$ (Phase 2), they, and robots that skipped Phases 1 and 2, (i.e. initially neither beacons nor corresponding to a beacon) will move along their unique lines to distance $\frac{s}{3}$ from $c$ (Phase 3) in $O(1)$ epochs in Algorithm $\{V, S, L\}$.

\textbf{Proof:} As previously shown, non-beacons are in position, their corresponding beacon is colored Beacon2, and the beacon’s lock and non-beacons are all visible to it after completing Phase 2.

Within 1 epoch, the non-beacons colored Lone1 see that their beacon is colored Beacon2 and change their color to Lone2.

Once a robot is colored Lone2, it moves along its current line $\ell_i$ until $d(r_i, c) = \frac{s}{3}$ in 1 epoch, unless it is already there, no matter what. This completes Phase 3 for non-beacons.

Consider a non-beacon robot $r_i$ on line $\ell_i$ colored Lone1 and its corresponding
beacon $r_b$ on line $\ell_b$. Only robots colored $\text{Lone1}$ or $\text{Lone2}$ lie between $\ell_i$ and $\ell_b$. The third condition for a beacon’s position in Phase 1 is such that non-beacons will move towards it and be strictly farther than $\frac{s}{3}$ from $c$. In the Phase 3, when robots move to exactly $d(r_i, c) = \frac{s}{3}$, this guarantees that all robots colored $\text{Lone2}$ move towards $c$ along their current lines. Non-beacon robots that move to lines closer to $\ell_b$ also are closer to $c$. This ensures that a robot colored $\text{Lone2}$ moving towards $c$ along its line will not cross a line segment between any robot colored $\text{Lone1}$ and its beacon. Therefore, a robot colored $\text{Lone1}$ cannot have its view of its beacon robot obstructed by a moving robot colored $\text{Lone2}$.

Once all visible robots between the beacon colored $\text{Beacon2}$ and its lock are colored $\text{Lone2}$ and at distance $\frac{s}{3}$ from $c$, the beacon changes its color to $\text{Beacon3}$ in 1 epoch. If a robot is still colored $\text{Lone1}$ then it must be visible to the beacon, as previously shown.

The locking robot then sees a robot on a subsequent line with the same distance from $c$ colored $\text{Beacon3}$ with only robots colored $\text{Lone2}$ at a distance $\frac{s}{3}$ from $c$ between them. The lock can see its beacon since otherwise would imply that the beacon could not see the lock initially. The lock then moves a distance $\frac{s}{3}$ from $c$ in 1 epoch. This completes Phase 3 for a lock.

The beacon colored $\text{Beacon3}$ on $\ell_i$ can finally move to $\frac{s}{3}$ from $c$ once its non-beacons and lock have moved. Recall the beacon moves halfway between adjacent lines in Phase 1. Therefore $\angle(\ell_i, c, \ell_{i+1}) = \angle(\ell_l, c, \ell_i)$, where $\ell_l$ is the line containing the associated locking robot. The beacon can use this to determine the line that should contain the locking robot whether or not it is visible. If there is no robot on $\ell_l$ with the same distance from $c$ colored $\text{Lock}$, then the locking robot must have already begun moving to $\frac{s}{3}$ from $c$,
and then so does the beacon. The lock then moves a distance $\frac{2}{3}$ from $c$ in 1 epoch. This completes Phase 3 for a beacon.

Lastly, there are the remaining robots that were initially neither beacons nor had a beacon. Specifically, robots on a unique line, whose previous line contained only a single robot as well. This robots do not executes Phases 1 or 2, since they have already achieved unique lines and do not need to guide others to unique lines. These robots begin Phase 3 colored Initial. They then satisfy Condition I5, and change their color to Lone2 in 1 epoch. They then complete Phase 3 as the previously discussed robots colored Lone2. □

**Lemma 6.8** No robot in Phase 3 collides with another robot in Phases 1, 2, or 3.

**Proof:** A robot in Phase 3 moves strictly along its unique line from $c$. Beacon robots in Phase 1 move either along their own unique line, or between two adjacent lines without reaching the neighboring line. Therefore a robot in Phase 3 cannot collide with a robot in Phase 1. Robots in Phase 2 do not move towards previous lines. Further, a non-beacon robot will not move in Phase 3 until all other non-beacons with the same beacon have reached their position in Phase 2, which is signalled by the beacon. Therefore a robot in Phase 3 cannot collide with a robot in Phase 2.

Two robots in Phase 3 are already on unique lines, which they do not move off of in this phase. Therefore, a robot in Phase 3 cannot collide with another robot that is in Phase 3. □

**Lemma 6.9** Once a robot only sees robots in the doorway or at $\frac{8}{3}$ from $c$, it moves through the door (Phase 4) in $O(1)$ epochs in Algorithm $\{\overline{V}, \overline{S}, L\}$.

**Proof:** Once a robot colored Lock, Beacon3, or Lone2 is at $\frac{8}{3}$ from $c$ along a unique line, and so are its neighbors, it changes it’s color to Final in 1 epoch. Every robot achieves
this by completing Phase 3, and each robot completes Phase 3 with one of the three afore-
mentioned colors, as previously shown.

When all robots have just colored themselves \textbf{Final}, robots are all on a circular arc
within the height of the door. This means they are all at unique heights and can there-
fore travel directly rightwards through the door. Robots on a circular arc also have mutual
visibility. Therefore, all robots must be ready to move rightwards through the door if all
visible robots on the arc are colored \textbf{Final}. Otherwise, there would have to be at least
one robot on the arc that isn’t colored \textbf{Final} that is adjacent to a robot that isn’t ready
to move through the door. That is to say, no robot moves in Phase 4 until all robots are
ready to move in Phase 4.

Once the robots are all colored \textbf{Final}, they move rightwards through the doorway
but do not disappear. If a robot \( r_i \) sees any robot \( r_j \) colored \textbf{Final} at distance \( d(c, r_j) < \frac{s}{3} \),
then conditions to move rightwards through the doorway have clearly been satisfied al-
ready. Thus, they move through the door in 1 epochs once all robots are colored \textbf{Final}.

Finally, once all visible robots are in the doorway, which implies that all robots are
in the doorway, robots disappear and terminate in 1 epoch.

\textbf{Corollary 6.10} A robot in Phase 4 can move through the door without causing a robot in
a previous phase to mistake its neighboring line, beacon, or lock.

\textbf{Lemma 6.11} No robot in Phase 4 collides with another robot.

\textbf{Proof:} Robots moving in Phase 4 cannot collide with a robot in Phases 1, 2, or 3 because
no robot moves in Phase 4 until all robots are ready to move in Phase 4, as shown in the
proof of Lemma 6.9.

Two robots moving in Phase 4 cannot collide since they start the phase on the
circumference of the top left quadrant of a circle and move strictly horizontally. That is,
robots in Phase 4 start with unique heights, and maintain that height as they move.

**Theorem 6.12** Algorithm $\{V, S, L\}$ solves Doorway Egress in $O(1)$ epochs without
 collisions using 10 colors.

6.7. Transparent Robots in ASYNC

In the classical model, robots are transparent. Therefore each robot can see the positions of all robots, and this guarantees knowledge of $n$. However, robots in this model have no persistent memory and no direct method of communication. Only positional information shared between robots. The setting of the robots and the terminology used will be the same as that of Section 6.6.

The algorithm of this section will use many concepts seen in Algorithm $\{V, S, L\}$, but there are small yet differences in many of the details.

6.7.1. Phase 0 - Position Locking Robots

The goal of the first phase is to place special robots called *locking robots*, or just locks. These locks help robots in subsequent phases determine when robots are no longer moving.

The innermost robot, or the only robot, on each line from the center of the doorway $c$ moves along its current line until it is a distance $\frac{2}{3}$ from $c$ or until it is tied for closest to $c$, whichever is closer to $c$. This is illustrated in Figure 6.8.
6.7.2. Phase 1 - Position Beacons

Next, special robots called *beacons* move into position to guide robots that are neither locks nor beacons to unique lines from $c$ in the next phase. Each beacon on line $\ell_i$ is associated with the closest lock on line $\ell_j$ such that $i \geq j$. A beacon will not move into place until its corresponding lock is in locking position.

There are two cases for a robot to become a beacon once in Phase 1:

1. A robot $r_i$ is on line $\ell_i$ with at least one other robot, $\ell_{i+1}$ has multiple robots or does not exist, and $r_i$ is furthest from $c$ on $\ell_i$.

2. A robot $r_i$ is on line $\ell_i$ with no other robot and $\ell_{i-1}$ has multiple robots.

The two cases account for either possibility regarding a line containing multiple robots, i.e., either the subsequent line $\ell_{i+1}$ contains exactly one robot, or it does not. That is, the next line may have multiple robots or there may be no subsequent line containing robots. If there is only one, then that’s the beacon. Otherwise, move a single robot out between $\ell_i$ and $\ell_{i+1}$ and then that robot falls into the previous case. In either situation, the beacon robot moves to the closest point satisfying the following conditions:

- It is on a line $\ell_k$ such that $\angle(\ell_{k-1}, c, \ell_k) = \angle(\ell_k, c, \ell_{k+1})$. That is, it’s line bisects the angle between adjacent lines.
- It is farthest or tied for farthest from $c$ out of all robots.
If line $\ell_i$ is the closest line to the half-plane line, then treat the half-plane line as $\ell_{i+1}$. Since the beacon is strictly moving in the empty regions between the adjacent lines of its initial configuration, the beacon moves into its position in one LCM cycle. An illustration of Phase 1 is provided in Figure 6.9.

![Figure 6.9. Beacon Robots Moving to Position](image)

6.7.3. Phase 2 - Move Non-Beacons to Unique Lines

The goal of this phase is to move the robots on a line containing multiple robots towards the beacon to unique lines from $c$. The locking robot need not move in this phase since all other robots will move off the line.

Before these robots move, if the outermost robot on $\ell_i$ has the same distance from $c$ as the beacon, it moves along $\ell_i$ halfway to the closest adjacent robot on $\ell_i$. This ensures that all of the robots on $\ell_i$ are closer to $c$ than the beacon.

With the beacon $r_b$ on line $\ell_{i+1}$ and locking robot on line $\ell_i$ both already in place, the remaining robots on $\ell_i$ move into the empty region between $\ell_i$ and $\ell_{i+1}$. These non-beacon robots move from $\ell_i$ to a new line $\ell_i'$ along the line segment between itself and the
beacon such that $\angle(\ell_i, c, \ell_i')$ increases as $d(r_i, c)$ decreases. Specifically, let $\theta' = \angle(\ell_i, c, \ell_i')$ and $\theta = \angle(\ell_i, c, \ell_{i+1})$. Then, each non-beacon robot moves such that $\theta' = \left(1 - \frac{d(r_i, c)}{d(r_{b}, c)}\right) \theta$.

Since a beacon robot moves in Phase 1 such that $d(r_i, c) \leq d(r_b, c)$, and the outermost non-beacon robot in Phase 2 ensures $d(r_i, c) < d(r_b, c)$, each non-beacon robot $r_i$ moves to a new line strictly between $\ell_i$ and $\ell_{i+1}$.

A robot $r_i$ keeps track of its corresponding beacon robot $r_b$, and thus $\ell_b$, throughout Phase 2 because the beacon is on the closest subsequent line containing on a single robot tied for furthest from $c$, and non-beacons move directly towards the beacon without reaching it (so they cannot be further from $c$ than the beacon). The described movement is illustrated in Figure 6.10.

![Figure 6.10. Movement of Non-Beacons Towards the Beacon](image-url)

The beacon, on line $\ell_b$, can see the closest line containing a locking robot, $\ell_l$, such that $l < b$. When all other robots are off of this line, the beacon knows that all of its corresponding non-beacons have moved or are moving to its current position. However, we want to ensure that the beacon waits until all robots have reached their final position.
to avoid any race conditions with the next step (e.g. the beacon has left beacon position to become a lock, and a corresponding non-beacon wakes up, moves along its line to become a lock, and collides with a non-beacon robot still moving very slowly or travelling far to its unique line). To avoid this issue, a beacon projects the robots positions back onto $\ell_l$ as shown in Figure 6.11 and determines the final position of each projection. Once all robots between the $\ell_b$ and $\ell_l$ are in position, the beacon knows all of its corresponding non-beacons are in place. It then moves to locking position.

![Figure 6.11. Projection of Non-Beacons Onto Their Initial Line](image)

**6.7.4. Phase 3 - Move Robots to Unique Heights In Front of Door**

The goal of this phase is to move all of the robots that have achieved unique lines in the prior phase to locking position. Recall that this means they will move to be tied for closest to $c$ or at $d(r_i, c) = \frac{s}{3}$, whichever is closer to $c$.

Once a robot $r_i$ on line $\ell_i$ wakes up and is on a unique line from $c$, if there is not a line $\ell_j$ such that $i < j$ containing a robot in beacon position before there is a line containing a only a lock, $r_i$ moves along $\ell_i$ to locking position.
The beacon and the lock have previously moved to locking position, and all of the non-beacons beacon are on unique lines, so the non-beacons can move along their current lines from $c$ to become locks in 1 epoch.

6.7.5. Phase 4 - Move Robots Through the Door

Once all robots are in locking position, they are on the circumference of one quadrant of a circle, as shown in Figure 6.12. This ensures that they are at unique heights and are able to move directly rightwards and reach the door.

When a robot $r_1$ is on $\ell_1$ by itself, and it sees that all other robots are in locking position (whether or not it is), just this robot moves through the doorway in a path that ensures the locking distance, the distance from $c$ corresponding to the current locking position, doesn’t change. To do this, $r_1$ moves along $\ell_1$ to the closest height such that a path rightwards through the door does not cross within locking distance of $c$, e.g. to a vertical distance of exactly $\frac{s}{3}$ from $c$. Then it moves rightward through the doorway without crossing any lines since it is already on $\ell_1$.

The other robots will then see a robot already in the doorway and they will move through it in 1 epoch. Once all robots are in the doorway, they disappear and terminate.
6.7.6. Algorithm \( \{V, S, L\} \)

The following condition-action pairs are written such that if multiple conditions are satisfied, the pair appearing first is chosen by the robot.

**Condition 1:** All robots are in the doorway

**Action 1:** Disappear and terminate

**Condition 2:** There is a robot in the doorway

**Action 2:** Move rightwards through the doorway

**Condition 3:** Robot \( r_i \) is on \( \ell_1 \) by itself, all other robots are on unique lines in locking position, and \( r_i \) cannot travel directly rightwards through the door without crossing within locking distance of \( c \) or hitting the wall

**Action 3:** Move along \( \ell_1 \) to the closest height such that a path rightwards through the door does not cross within locking distance of \( c \)
Condition 4: Robot $r_i$ is on $\ell_1$ by itself, all other robots are on unique lines in locking position, and $r_i$ can travel directly rightwards through the door without crossing within locking distance of $c$ or hitting the wall

**Action 4:** Move directly rightwards through the door

Condition 5: Robot $r_i$ is on a unique line from $c$, there is not a line $\ell_j$ such that $i < j$ containing a robot in beacon position before there is a line containing a only a lock, and $r_i$ is not in locking position

**Action 5:** Move along current line to locking position. That is, closest (or tied for closest) to $c$ or at $d(r_i, c) = \frac{s}{3}$, whichever is closer to $c$.

Condition 6: Robot $r_i$ is in beacon position (on a line that bisects the angle between neighboring lines containing locking robots, no robot is further from $c$), all robots between $r_i$ and the closest line containing a lock, $\ell_l$ such that $l < i$, are in position (which is determined using projection).

**Action 6:** Move along current line to locking position. That is, closest (or tied for closest) to $c$ or at $d(r_i, c) = \frac{s}{3}$, whichever is closer to $c$.

Condition 7: Robot $r_i$ is on a line $\ell_i$ with multiple robots, the innermost robot $r_j$ on the line is in locking position, $i \neq j$, the line before the next line containing a locking robot contains only a robot in beacon position, and every robot on $\ell_i$ is closer to $c$ than the beacon.

**Action 7:** Move directly towards the beacon to a unique line creating the proper angle from the current line, i.e. $\theta' = \left(1 - \frac{d(r_i, c)}{d(r_b, c)}\right) \theta$. 105
Condition 8: Robot $r_i$ is on a line with multiple robots, is farthest from $c$ on its line, and the next line contains only a robot in beacon position.

Action 8: Move along current line halfway to the closest robot on the line.

Condition 9: Robot $r_i$ is on a unique line from $c$, the previous line has multiple robots, the innermost robot on the previous line is in locking position, and $r_i$ is not in beacon position ($r_i$’s line bisects the angle between adjacent lines and no robot is farther than $r_i$ from $c$)

Action 9: Move to the closest point on a new line that bisects the angle between adjacent lines $\ell_i$ and $\ell_{i+1}$ such that no robot is farther from $c$ than $r_i$. I.e. move to beacon position.

Condition 10: Robot $r_i$ is on a line with multiple robots, it is the furthest from $c$ on its current line, the innermost robot on the current line is in locking position, and either the next line has multiple robots or there is no subsequent line containing robots.

Action 10: Move to the closest point on a new line that bisects the angle between adjacent lines $\ell_i$ and $\ell_{i+1}$ such that no robot is farther from $c$ than $r_i$. I.e. move to beacon position.

Condition 11: Robot $r_i$ is the closest robot to $c$ (or the only robot) on its line, and is not in locking position; i.e., closest (or tied for closest) to $c$ or at $d(r_i, c) = \frac{s}{3}$, whichever is closer to $c$.

Action 11: Move along current line to locking position

6.7.7. Analysis

Lemma 6.13 Robots can always determine whether they are a beacon, and non-beacons can always determine their corresponding beacon.

Proof: There are two cases for a robot to become a beacon:
1. A robot \( r_i \) is on line \( \ell_i \) with at least one other robot, \( \ell_{i+1} \) has multiple robots or does not exist, and \( r_i \) is furthest from \( c \) on \( \ell_i \).

2. A robot \( r_i \) is on line \( \ell_i \) with no other robot and \( \ell_{i-1} \) has multiple robots.

The criteria for initially determining a beacon for each line \( \ell_i \) having multiple robots only depends on knowledge of: the position of robots on adjacent lines, if a robot is the farthest robot from \( c \) on \( \ell_i \), and if there are multiple robots on \( \ell_i \). Robots have transparency, thus each of the robots have knowledge of these criteria at any given time.

A robot \( r_i \) keeps track of it’s corresponding beacon robot \( r_b \), and thus \( \ell_b \), throughout Phase 2 because the beacon is on the closest subsequent line containing on a single robot tied for furthest from \( c \), and non-beacons move directly towards the beacon without reaching it (so they cannot be further from \( c \) than the beacon).

In subsequent phases, a beacon is no longer used.

\[ \square \]

**Lemma 6.14**  *Beacon and lock move into position (Phases 0 and 1) in \( O(1) \) epochs in Algorithm \( \{V,S,L\} \).*

**Proof:** The innermost robot on each line moves along its current line to a distance of \( \frac{s}{3} \) from \( c \) or until it is tied for closest to \( c \), whichever is closer to \( c \). This guarantees each locking robot will move along unique lines from each other and, since they move no further from \( c \), they move along a path where no other robots can be (otherwise they are not innermost). Since there are no conditions to satisfy aside from being innermost, they move in 1 epoch.

Meanwhile, a beacon robot \( r_b \) on line \( \ell_i \) in moves into the empty space between \( \ell_i \) and \( \ell_{i+1} \). As previously shown, there is exactly one beacon for each initial line having multiple robot which satisfies either Condition 9 or 10 initially. Corresponding non-beacons do
not move off their line until the beacon is in place. Therefore, each beacon moves without waiting in 1 epoch.

\begin{lemma}
No robot in Phase 0 or 1 collides with another robot in Phase 0 or 1.
\end{lemma}

\begin{proof}
The innermost robot on each line moves inward along its current line in Phase 0, if at all. This guarantees each locking robot will move along unique lines from each other and, since they move no further from \( c \), they move along a unique path (otherwise they are not innermost).

Recall the two cases for a robot to become a beacon. The beacons in Case 1 move within empty, non-overlapping regions. That is, there is one beacon robot for line \( \ell_i \), which moves strictly between \( \ell_i \) and \( \ell_{i+1} \). A collision between beacons would imply that one line has two beacons, resulting in a contradiction since there cannot be two outermost robots on \( \ell_i \). The beacons in Case 2 move strictly between adjacent lines. By the same logic as that of Case 1, no other beacons enter this region. Only beacon robots and locks move in Phase 1.
\end{proof}

\begin{lemma}
Once the corresponding beacon and lock are in place (Phases 0 and 1), corresponding non-beacons move to form unique lines from \( c \) (Phase 2) in \( O(1) \) epochs in Algorithm \( \{V, S, L\} \).
\end{lemma}

\begin{proof}
First, if the outermost robot on \( \ell_i \) has the same distance from \( c \) as the beacon, it moves along \( \ell_i \) halfway to the closest adjacent robot on \( \ell_i \). This ensures that all of the robots on \( \ell_i \) are closer to \( c \) than the beacon, since the beacon moved to be tied for furthest from \( c \). This outermost non-beacon satisfies Condition 8 in this case and moves in 1 epoch. Otherwise, this is not needed and the phase starts with the following.

The non-beacons see that all robots on their line are closer to \( c \) than their beacon,
the innermost robot on their line is in locking position, and the next line is a robot in beacon position, i.e. halfway between adjacent lines and no robot is farther from c. This indicates that they are free to move towards it.

Robots on \( \ell_i \) change move directly towards the beacon. Even without transparency, these robots all move along non-overlapping lines towards the beacon robot, so no robot can have its view of the beacon obstructed by a robot moving towards it. Further, even as non-beacons move, the beacon is known to all of its non-beacons as shown in the proof of Lemma 6.13. To get each robot to a unique line from c, non-beacons map their distance from c to a percentage of the angle between \( \ell_i \) and \( \ell_b \), the line containing the beacon. Since the beacon is guaranteed to be further from the center than any moving non-beacon, the Euclidean distance \( d(r_b, c) > d(r_i, c) \) for any robot \( r_i \) on \( \ell_i \). Further, given that \( d(r_i, c) \) is unique for all \( r_i \) on \( \ell_i \), they all move to unique line between \( \ell_i \) and \( \ell_b \). Therefore, each non-beacon determines a unique angle and moves to a unique line from c in 1 epoch once the non-beacons are free to move.

No other robot sharing the same beacon will move to the same line from c after Phase 2 since that would imply the same distance from c along \( \ell_i \), i.e., it would imply the same initial position. \( \square \)

**Lemma 6.17** No robot in Phase 2 collides with another robot in Phases 0, 1, or 2.

**Proof:** The non-beacon robots move along unique, non-overlapping paths towards the beacon. Thus, non-beacons do not collide with each other.

The beacon and lock do not move in this phase. Thus, they do not collide with any robot, unless another robot moves to their position.
Recall that each non-beacon robot moves such that $\theta' = \left(1 - \frac{d(r_i, c)}{d(r_b, c)} \right) \theta$. Since a non-beacon $r_j$ satisfies $d(r_j, c) < d(r_b, c)$ due to the movement of $r_b$ in Phase 1, each non-beacon robot $r_i$ moves to a new line strictly between $\ell_i$ and $\ell_b$. Thus, the non-beacons do not collide with the beacon or lock.

No collisions occur with robots in previous phases since only beacons and locks move in Phases 0 and 1, and if a beacon were in the region between $\ell_i$ and $\ell_b$, that would contradict the assumption that $r_b$ is the beacon for robots on $\ell_i$. The same logic can be used to explain why there are no non-beacons that corresponding to a different beacon that collide with the $r_b$ or its non-beacons. That is, $r_b$ and its corresponding non-beacons and lock will not collide with robots outside of this set of robots.

**Lemma 6.18** Once a beacon and its corresponding non-beacon robots and lock are on unique lines from $c$ (Phase 2), they, and robots that skipped Phases 1 and 2, (i.e. initially neither beacons nor corresponding to a beacon) will move along their unique lines to distance $\frac{s}{3}$ from $c$ (Phase 3) in $O(1)$ epochs in Algorithm \{V, S, L\}.

**Proof:** The beacon will not move until it sees that all of it’s non-beacons are in place. As previously this occurs in Phase 2 in $O(1)$ epochs. The beacon then projects the robots positions back onto $\ell_l$, the closest line containing a lock, as shown in Figure 6.11 and determines the final position of each projection. Once all robots between the $\ell_b$ and $\ell_l$ are in position, the beacon knows all of its corresponding non-beacons are in place. It then moves to locking position in 1 epoch.

Recall that the lock is already in locking position. Now that the beacon is too, the corresponding non-beacons lie between two locks without a beacon between.
Subsequently, the non-beacons satisfy Condition 5 since there is a line with a locking robot closer than a line with a beacon on each side. They do no satisfy any higher priority conditions, causing them to operate incorrectly, because that would imply that all robots are already in Phase 4. This cannot be true since the non-beacons are strictly farther from $c$ than locks. Therefore, the non-beacons move to locking position in 1 epoch once the beacon has moved.

Lemma 6.19 No robot in Phase 3 collides with another robot in Phases 0, 1, 2, or 3.

Proof: A robot in Phase 3 moves strictly along its unique line from $c$. Beacons and locking robots in Phases 0 or 1 move either along their own unique line, or between two adjacent lines without reaching the neighboring line. Therefore a robot in Phase 3 cannot collide with a robot in Phases 0 or 1. The beacon does not move off its line in Phase 2, and robots in Phase 3 move strictly along their own unique lines. Therefore a robot in Phase 3 cannot collide with a robot in Phases 2 or 3.

Lemma 6.20 Once all robots are in locking position, they move through the door (Phase 4) in $O(1)$ epochs in Algorithm $\{V, S, L\}$.

Proof: Phase 4 starts with all robots in locking position. That is, they are all at unique heights along the circumference of one quadrant of a circle. Since all robots are visible, no robot moves in Phase 4 until all robots are in Phase 4.

The robot $r_1$ on the unique line closest to the wall $\ell_1$ sees all other robots are in locking position. Regardless of whether it is in locking position, $r_1$ checks to see if it can move rightward through the doorway without crossing within locking distance of $c$ during the movement. If it cannot, it satisfies Condition 3 and moves to the closest point along its line such that it can travel directly rightward through the doorway without crossing.
within locking distance. Since its locking distance from \( c \) is no farther than \( \frac{2}{3} \), there is some such point. It moves to this point in 1 epoch.

Robot \( r_1 \) then satisfies Condition 4 and travels rightwards through the door in 1 epoch.

Condition 1 is not met, but all robots see that there is now a robot in the doorway which satisfies Condition 2. It doesn’t matter if other conditions are satisfied as well since lower-numbered conditions are selected first. Since the remaining robots started the phase at unique heights in front of the door, they can then move directly rightwards through the door in 1 epoch.

Then Condition 1 satisfies and all robots disappear and terminate in 1 epoch. □

**Corollary 6.21** A robot in Phase 4 can move through the door without causing a robot in a previous phase to mistake its neighboring line, beacon, or lock. □

**Lemma 6.22** No robot in Phase 4 collides with another robot.

**Proof:** Robots moving in Phase 4 cannot collide with a robot in Phases 0, 1, 2, or 3 because no robot moves in Phase 4 until all robots are ready to move in Phase 4, as shown in the proof of Lemma 6.20.

The robot \( r_1 \) moves strictly outside of the circumference of the circle until reaching the door, and other robots cannot move until it has reached the door. Therefore \( r_1 \) does not collide with any robot in Phase 4.

No two of the remaining robots moving in Phase 4 can collide since they start the phase on the circumference of the top left quadrant of a circle and move strictly horizontally. That is, robots in Phase 4 start with unique heights, and maintain that height as they move. □
Theorem 6.23 Algorithm \( \{V, S, L\} \) solves DOORWAY EGRESS in \( O(1) \) epochs without collisions.

6.8. Opaque Robots in SSYNC

In the robots with lights model, robots are typically opaque. However, their lights allow them a form of communication and persistent memory to compensate for the lack of global visibility. These algorithms often seek to minimize the palette size of the robots’ persistent memory (number of colors). We have provided an asynchronous algorithm for robots with lights having 10 colors in Section 6.6. This section provides a semi-synchronous algorithm for the extreme case of only a single color in the robots with lights model, i.e. effectively the same as having the lights always set to off. Without the ability to see every robot, and without the ability to communicate or retain memory across cycles, we show that a scheduler enforcing some level of synchrony (namely, SSYNC) is sufficient.

Once again, some details are the same as either/both of the previous algorithms; however, there are crucial distinctions in the details. For ease of reading, we briefly restate some details in this section in lieu of referring to prior sections in several places.

6.8.1. Phase 1 - Position Beacons

Once again, the goal of the first phase is to position the beacons. Beacons will be used in the following phase to lead their corresponding non-beacons to unique lines from the center of the doorway \( c \).

There are two cases for a robot to become a beacon:

1. A robot \( r_i \) is on line \( \ell_i \) with at least one other robot, \( \ell_{i+1} \) has multiple robots or does not exist, and \( r_i \) is furthest from \( c \) on \( \ell_i \).
2. A robot \( r_i \) is on line \( \ell_i \) with no other robot and \( \ell_{i-1} \) has multiple robots.

The two cases account for either possibility regarding a line containing multiple robots, i.e., either the subsequent line \( \ell_{i+1} \) contains exactly one robot, or it does not. That is, the next line may have multiple robots or there may be no subsequent line containing robots. If there is only one, then that’s the beacon. Otherwise, move a single robot out between \( \ell_i \) and \( \ell_{i+1} \) and then that robot falls into the previous case.

In the first case, the beacon robot moves to the closest point satisfying the following conditions:

- It is on a line \( \ell_k \) such that \( \angle(\ell_{k-1}, c, \ell_k) = \angle(\ell_k, c, \ell_{k+1}) \). That is, it’s line bisects the angle between adjacent lines.
- It is farther from \( c \) than robots on the previous line.
- It is no closer to \( c \) at any point during or after the movement.

Moving the beacon to a new line halfway between adjacent lines is not compulsory in this algorithm, so long as the robot moves to a unique line and does not cross any lines. This is because no robot can be seen in motion in SSYNC. Therefore, the beacon is seen in its initial position until it is in beacon position. For this same reason, no locking robot is needed. In case two, the robot is already on a unique line, so it need not move to be recognized as a beacon.

If line \( \ell_i \) is the closest line to the half-plane line, then treat the half-plane line as \( \ell_{i+1} \). Since the beacon is strictly moving in the empty regions between the adjacent lines of its initial configuration, the beacon moves into its position in one LCM cycle. An illustration of Phase 1 is provided in Figure 6.13.
6.8.2. Phase 2 - Move Non-Beacons to Unique Lines

The goal of this phase is to move the robots on a line containing multiple robots towards the beacon to unique lines from \( c \). The robot closest to \( c \) on the line with multiple robots need not move in this phase since all other robots will move off the line.

With only a single robot \( r_b \) on line \( \ell_{i+1} \), any robot \( r_j \) on \( \ell_i \) will move into the empty region between \( \ell_i \) and \( \ell_{i+1} \). These non-beacon robots move from \( \ell_i \) to a new line \( \ell'_i \) along the line segment between itself and the beacon such that \( \angle(\ell_i, c, \ell'_i) \) increases as \( d(r_i, c) \) decreases. Specifically, let \( \theta' = \angle(\ell_i, c, \ell'_i) \) and \( \theta = \angle(\ell_i, c, \ell_{i+1}) \). Then, each non-beacon robot moves such that \( \theta' = \left( 1 - \frac{d(r_i, c)}{d(r_i, c) + 1} \right) \theta \).

Since \( 0 < \theta' < \theta \), each non-beacon robot \( r_i \) moves to a new line strictly between \( \ell_i \) and \( \ell_{i+1} \).

Note that, in this algorithm, only a subset of the non-beacon robots may be activated during an LCM cycle. These robots will move at the same time towards their common beacon. The previously uppermost non-beacon on \( \ell_i \) that moved (which is the closest
to $\ell_i$ after the move) will then become the beacon for the remaining non-beacons on $\ell_i$.

These remaining non-beacons are activated within one epoch, and can immediately move to a beacon, if not the original. Since a robot is never seen in motion in SSYNC, the beacon is always seen as a single robot on the subsequent line with no robots in between lines. The described movement is illustrated in Figure 6.14.

![Figure 6.14. Movement of Non-Beacons Towards the Beacon](image)

Since no robots are seen in motion, as soon as the beacon $r_b$ wakes up and sees only a single robot $r'_b$ on the previous line, $r_b$ will never again act as a beacon. That is, $r'_b$ will become the new beacon for the non-beacons previously corresponding to $r_b$ in the event that some of the other non-beacons have yet to move.

### 6.8.3. Phase 3 - Move Robots to Unique Heights In Front of Door

The goal of this phase is to move all of the robots that have achieved unique lines in the prior phase to a common distance from $c$ that would allow all robots to travel horizontally through the door.

Once a robot $r_i$ on line $\ell_i$ wakes up and is on a unique line from $c$, and both $\ell_{i-1}$
and $\ell_{i+1}$ contain only a single robot, $r_i$ knows that it will not be needed to serve as a beacon at any point thereon. It then moves along its current line $\ell_i$ until $d(r_i, c) = \frac{s}{3}$. Since the line is unique and robots in prior phases do not cross lines from $c$, this occurs in 1 epoch without collisions once the aforementioned conditions are satisfied.

6.8.4. Phase 4 - Move Robots Through the Door

Once all robots have completed the prior phase, they are on the circumference of one quadrant of a circle, as shown in Figure 6.15. This ensures that they are at unique heights and are able to move directly rightwards and reach the door. However, a robot may execute Phase 4 when ready while another robot is in a prior phase.

When robots reach the doorway, their lines from $c$ effectively disappear. However, since the algorithm is semi-synchronous, robots are not seen while moving to the door. That is, they are seen on their proper line then, during the next LCM cycle, the line is gone. The conditions for a robot moving to the door ensure that the robot’s line is no longer relevant to any other robot, e.g. a non-beacon in Phase 2.

Figure 6.15. Robots Move Horizontally Through the Door
Once all visible robots are in the doorway, which implies all robots are in the doorway, the robots disappear and terminate.

6.8.5. Algorithm \( \{ V, S, L \} \)

The following condition-action pairs are written such that if multiple conditions are satisfied, the pair appearing first is chosen by the robot.

**Condition 1:** All visible robots are in the doorway.

**Action 1:** Disappear and terminate.

**Condition 2:** Robot \( r_i \) only sees robots in the doorway or at distance \( s_3 \) from \( c \).

**Action 2:** Move rightwards through the doorway.

**Condition 3:** Robot \( r_i \) is on a unique line from \( c \) and so are robots on neighboring lines, and \( d(r_i, c) \neq \frac{s_3}{3} \).

**Action 3:** Move along current line to a distance \( \frac{s_3}{3} \) from \( c \).

**Condition 4:** Robot \( r_i \) is on a line with multiple robots, and the next line contains a single robot.

**Action 4:** Move directly towards the beacon to a unique line creating the proper angle from the current line, i.e. 
\[
\theta' = \left( 1 - \frac{d(r_i, c)}{d(r_i, c) + 1} \right) \theta
\]

**Condition 5:** Robot \( r_i \) is on a line with multiple robots, it is the furthest from \( c \) on its current line, and either the next line has multiple robots or there is no subsequent line containing robots.

**Action 5:** Move to the closest point on a new line halfway between adjacent lines \( \ell_i \) and 118
such that \( r_i \) is father from \( c \) than any robot on \( \ell_i \), and \( r_i \) moves such that it is no closer to \( c \) during or after the movement.

6.8.6. Analysis

Lemma 6.24 Robots can always determine whether they are a beacon, and non-beacons can always determine their corresponding beacon.

Proof. There are two cases for a robot to become a beacon:

1. A robot \( r_i \) is on line \( \ell_i \) with at least one other robot, \( \ell_{i+1} \) has multiple robots or does not exist, and \( r_i \) is furthest from \( c \) on \( \ell_i \).

2. A robot \( r_i \) is on line \( \ell_i \) with no other robot and \( \ell_{i-1} \) has multiple robots.

The criteria for determining a beacon for each line \( \ell_i \) having multiple robots only depends on knowledge of: adjacent lines and if there are multiple robots on \( \ell_i \). A robot can see every robot on its adjacent lines, since there can be no robots on a line between them to obstruct their visibility. All robots know the location of \( c \), so robots know their line and thus can see if they are on it by themselves and if they are the outermost robot on their line. □

Lemma 6.25 Beacons move into position (Phase 1) in \( O(1) \) epochs in Algorithm \( \{\overline{V}, S, \overline{L}\} \).

Proof. Recall the two cases for a robot to be designated a beacon stated in Lemma 6.24.

A beacon robot \( r_b \) on line \( \ell_i \) in Case 1 moves into the empty space between \( \ell_i \) and \( \ell_{i+1} \) without waiting in 1 epoch.

A beacon robot \( r_b \) on line \( \ell_i \) in Case 2 need not move since it is already on a unique line, and therefore the non-beacon robots on the prior line, if it contains multiple robots, will satisfy Condition 4.
In Algorithm \( \{V, S, L\} \), the phases can overlap. That is, only a subset of the non-beacon robots may be activated during an LCM cycle. These robots will move at the same time towards their common beacon. The previously uppermost non-beacon on \( \ell_i \) that moved (which is the closest to \( \ell_i \) after the move) will then become the beacon for the remaining non-beacons on \( \ell_i \). These remaining non-beacons are activated within 1 epoch, and can immediately move to a beacon, if not the original. Thus, a beacon will be in place for all of the non-beacons originally on the same line within 1 epoch.

**Lemma 6.26** No robot in Phase 1 collides with another robot in Phase 1.

**Proof:** The beacons in Case 1 move within empty, non-overlapping regions. That is, there is one beacon robot for line \( \ell_i \), which moves strictly between \( \ell_i \) and \( \ell_{i+1} \). A collision between beacons would imply that one line has two beacons. The beacons in Case 2 do not move. Only beacon robots move in Phase 1.

**Lemma 6.27** Once their beacon has moved, corresponding non-beacons move to form unique lines from \( c \) (Phase 2) in \( O(1) \) epochs in Algorithm \( \{V, S, L\} \).

**Proof:** The non-beacons will know they are free to move when there is a single robot on the subsequent line. This is guaranteed as soon as the beacon has completed Phase 1. No robot can be seen while in motion due to SSYNC, so all non-beacons on \( \ell_i \) agree on their beacon’s position. To get each robot to a unique line from \( c \), they map their distance from \( c \) to a percentage of the angle between \( \ell_i \) and \( \ell_b \), the line containing the beacon. Since \( 0 < \theta' < \theta \), each non-beacon robot \( r_i \) moves to a new line strictly between \( \ell_i \) and \( \ell_{i+1} \).

Since at no point can a robot’s visibility of its beacon be obstructed, and it always knows its distance from \( c \), it determines a unique angle and moves to a unique line from \( c \) in 1 epoch once the non-beacons are free to move.
As previously described in the proof of Lemma 6.25, if Phase 1 and Phase 2 overlap, it will still take 1 epoch for a non-beacon to move towards a beacon, this robot becomes a beacon implicitly, and the remaining non-beacons move towards the new beacon to unique lines.

\[ \square \]

**Lemma 6.28** No robot in Phase 2 collides with another robot in Phases 1 or 2.

**Proof:** The non-beacon robots move along unique, non-overlapping paths towards the beacon. Thus, non-beacons do not collide with each other. The beacon doesn’t move off its current line as non-beacons move towards it. The non-beacons will not move towards a beacon in motion, and since the algorithm is semi-synchronous, the beacon cannot be seen in transit. Recall that each non-beacon robot moves such that \( \theta' = \left(1 - \frac{d(r_i, c)}{d(r_i, c) + 1}\right) \theta \). Since \( 0 < \theta' < \theta \), each non-beacon robot \( r_i \) moves to a new line strictly between \( \ell_i \) and \( \ell_{i+1} \).

Thus the non-beacons do not collide with the beacon.

**Lemma 6.29** Once a robot and its neighbors are on unique lines from \( c \), the robot will move along its current line to distance \( \frac{s}{3} \) from \( c \) (Phase 3) in \( O(1) \) epochs in Algorithm \( \{\mathcal{V}, S, L\} \).

**Proof:** Once a robot has completed, or skipped, Phases 1 and 2, it will be on a unique line. A robot will ensure that it doesn’t become a beacon before moving to \( \frac{s}{3} \) from \( c \) by waiting for its neighbors to be on unique lines as well. Thus, once it and its neighbors have completed Phases 1 and 2 (in \( O(1) \) epochs as previously shown), it moves directly along it’s current line to a distance \( \frac{s}{3} \) from \( c \) without waiting in 1 epoch.

**Lemma 6.30** No robot in Phase 3 collides with another robot in Phases 1, 2, or 3.

**Proof:** A robot in Phase 3 moves strictly along its unique line from \( c \). Beacon robots in Phase 1 move between two adjacent lines without reaching the neighboring line. There-
before a robot in Phase 3 cannot collide with a robot in Phase 1. A robot in Phase 3 is not a beacon, since the previous line has only one robot, therefore no robot in Phase 2 can move within the region between $\ell_i$ and $\ell_{i-1}$. Robots in Phase 2 also do not move towards previous lines. Therefore a robot in Phase 3 cannot collide with a robot in Phase 2.

**Lemma 6.31** Once a robot only sees robots in the doorway or at $\frac{s}{3}$ from c, it moves through the door (Phase 4) in $O(1)$ epochs in Algorithm $\{V, S, L\}$.

**Proof:** A robot in Phase 4 waits until it sees only robots that have completed Phase 3 to move. Robots complete the previous phases within $O(1)$ epochs as previously shown. Robots in Phase 4 are positioned on unique lines at exactly $\frac{s}{3}$ from c. Therefore, in 1 epoch, robots can move rightwards through the door without waiting once all robots are in Phase 4.

**Lemma 6.32** A robot in Phase 4 can move through the door without disrupting previous phases.

**Proof:** A robot $r_i$ in Phase 4 may be obstructed from seeing that there is a robot in a previous phase and move through the door. Since the algorithm is semi-synchronous, robots are not seen while moving towards the doorway. That is, they are seen on their line then, during the next LCM cycle, the line is gone. The conditions for a robot moving through the door in Phase 4 ensure that the robot’s line is no longer relevant to any other robot, e.g. a non-beacon in Phase 2. This is because:

- Since $r_i$'s prior line contains only a single robot, there are no non-beacons that can use $r_i$ as a beacon. Thus, there is no conflict with any robot in Phases 1 or 2.

- A robot $r_j$ meeting the conditions for Phase 3 will have a new adjacent line after $r_i$ moves. If the new adjacent line contains only one robot, nothing changes and $r_j$ still meets the conditions for Phase 3 and moves along its current line to $d(r_j, c) = \frac{s}{3}$. Otherwise, it waits for the new adjacent line to contain only one robot, which
becomes true for all robots in $O(1)$ epochs as previously shown.

\textbf{Lemma 6.33} No robot in Phase 4 collides with another robot.

\textbf{Proof}: When a robot $r_i$ is on $\ell_1$ by itself at $d(r_i, c) = \frac{s}{3}$, and so are all robots visible to it, it will move through the door. It will not collide with a robot $r_o$ in a prior phase because this would imply that $r_o$ satisfied $d(r_i, c) < \frac{s}{3}$ before the movement or a collision occurred due to a robot entering within $\frac{s}{3}$ during the movement. If the former were true, $r_i$ would have seen it. If the former is not true, then the latter cannot be true because:

\begin{itemize}
\item Beacons moving in Phase 1 move no closer to $c$.
\item A Phase 2 robot moving towards its beacon will not cross paths with a robot moving to the door in phase 4 because a beacon on $\ell_i$ moves towards a robot on line $\ell_{i+1}$. For a collision to occur, the robot moving in Phase 4 must have begun between $\ell_i$ and $\ell_{i+1}$ since the beacon and non-beacon were further from $c$ than $\frac{s}{3}$ before moving. If the Phase 4 robot were between these two lines, this contradicts the assumption that a robot on $\ell_i$ moved towards a robot on $\ell_{i+1}$.
\item A robot in Phase 3 will only move such that $d(r_i, c) = \frac{s}{3}$. So if it did not initially satisfy $d(r_i, c) < \frac{s}{3}$, it would not satisfy it during the movement.
\end{itemize}

Since robots move horizontally in Phase 4 and start the phase at unique heights, no two robots in Phase 4 collide.

\textbf{Theorem 6.34} Algorithm $\{\overline{V}, S, \overline{L}\}$ solves \textsc{Doorway Egress} in $O(1)$ epochs without collisions.

6.9. Handling Point Robots on the Half-Plane Line

Recall that the algorithms we have provided operate independently on each half-plane. Note that explicit CA pairs are not provided for the case of robots lying on the half-plane line in the main algorithms. In this section, we detail the procedure used in this edge-case to handle the situation where one or more robots don’t lie in either half-
plane initially. The approach is used after the main algorithm as needed and is the same for each of the three provided algorithms.

The robots on the half-plane line wait until the only remaining robots are on the half-plane line. Then, the closest robot to the wall moves to a position on the door outside of a position that any robot would move to in the main algorithm, i.e., farther from $c$ than $s_3$ on an arbitrary side. This robot doesn’t disappear once it reaches the door; this robot acts as a beacon to the robots on the middle line. The beacon informs the non-beacons that they are now executing the portion of the algorithm in which robots on the half-plane line move to the door. The remaining robots then map their current location on the half-plane line to a position on the door. Specifically, robots further from the wall on the half-plane line move closer to the wall on the door using $d(t_i, c) = s \left( \frac{d(r_i, c)}{3(d(r_i, c) + 1)} \right)$, where $t_i$ is the target position of robot $r_i$ on the door. The non-beacon robots move to the opposite side of the half-plane line to ensure that no moving robot obstructs the view of the beacon.

While this isn’t the only method that can be used (for example, in SSYNC, we can skip the step of sending a single robot to a special point on the door), this method works for each of our three algorithms. Further, it allows handling this edge-case to be a disjoint procedure appended to the end of the algorithm as needed, rather than complicating the primary algorithm. The procedure is illustrated in Figures 6.16 and 6.17.
6.10. Conclusion

In this chapter, we have introduced the problem of Doorway Egress and provided three optimal (in terms of time) algorithms for specific model variants and provided a linear-time algorithm for any model variant. However, the linear-time algorithm has the advantage of asymptotically optimal distance and spatial complexities.
It would be interesting to try to reduce the distance and spatial complexities of the time-optimal algorithms so that there is no need to prioritize either time or energy-efficiency and space. Finally, it would be of interest to characterize the relationship between the capabilities of the robots.
Chapter 7. Doorway Egress by Fat Robots

In this chapter, we continue to look at Doorway Egress, now in the context of fat robots. For fat robots, we assume that they are initially aligned to the points of an imaginary grid, though this is only required for the initial configuration. We call this grid-alignment. We further assume that \( s \geq 2 \). That is, the entire doorway is at least two units wide. This guarantees that the doorway has at least unit width on each half-plane, allowing sufficient room for the two half-planes to operate independently, as in the previous chapter. If the width of the doorway is at least 1 unit wide, but less than 2, then fat robots will use the simpler, linear-time algorithm as \( \Omega(n) \) time would be required. If the width of the doorway were less than 1 unit, the problem is unsolvable for fat robots.

7.1. Contributions of This Chapter

In this chapter, we establish two results (summarized in Table 7.1). We include the linear-time algorithm of the previous chapter, as it can also be applied to fat robots with grid-alignment, in Table 7.1 for simplicity.

**Results 1-2:** Two \( O(\sqrt{n} + \frac{n}{s}) \) time, and therefore, optimal, algorithms for fat robots with each algorithm making use of grid-alignment, transparency and exactly one of \{lights, SSYNC\}.

Both algorithms are shown to be collision-free. The ability set of this chapter is \{G, V, S, L\}. The set \{G, V, S, L\} represents the inclusion of grid-alignment, transparency, SSYNC, and lights.

7.2. Chapter Roadmap

We discuss the model variants and preliminaries in Section 7.3. Sections 7.4 and 7.5 describe Results 5-6 (Algorithms \{G, V, S, L\} and \{G, V, \overline{S}, L\}), respectively. Section 7.6
Table 7.1. Summary of DOORWAY EGRESS Algorithms for Fat Robots.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Extent</th>
<th>Colors</th>
<th>Visibility</th>
<th>Schedule</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Fat</td>
<td>−</td>
<td>Any</td>
<td>Any</td>
<td>$\Omega(\sqrt{n} + \frac{n}{s})$</td>
</tr>
<tr>
<td>${G, V, S, L}$</td>
<td>Fat</td>
<td>−</td>
<td>Obstructed</td>
<td>ASYNC</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>${G, V, S, L}$</td>
<td>Fat</td>
<td>9</td>
<td>Global</td>
<td>SSYNC</td>
<td>$O(\sqrt{n} + \frac{n}{s})$</td>
</tr>
</tbody>
</table>

then provides a brief discussion.

7.3. Preliminaries

In this section, we provide the details of the model variants used in this chapter. For information on the setting and problem definition, please refer to Section 6.4 in the previous chapter.

In this chapter, we consider fat robots in $\mathbb{R}^2$ with the real plane being divided by an infinite wall containing a door. The fat robots are discs of unit diameter. As in the point-robots model, fat robots are initially placed arbitrarily on one side of the wall in non-overlapping (even partially) positions. However, we assume that fat robots are initially grid-aligned. That is, the initial configuration of fat robots places each robot in a cell of an imaginary, infinite 2-dimensional discrete grid. The origin of the grid is at the center of the doorway, and the grid lines are parallel or perpendicular to the wall. Each cell is of unit width, i.e., the diameter of a robot. Note that placing all robots in grid cells is equivalent to placing robots at vertices of the grid that is shifted vertically and horizontally by half of one unit. The robots all agree on the position of the grid as the position of the doorway is always visible to transparent robots and each robot is of unit diameter.

We make this assumption as it seems that DOORWAY EGRESS is not easily solved without the assumption of grid-alignment. Further, aligning the robots to the grid quickly
(without increasing the bound on time) appears to be non-trivial and should be addressed in the future as a separate problem.

Robots’ movements, both for fat and point robots, are restricted to straight paths by design. The robots may be capable of moving along curved paths, but the algorithms would not make use of this ability. Fat robots are capable of moving any distance in any direction, just as the point robots are. Therefore, they are not restricted to movements along the edges of the grid or movements that preserve grid-alignment.

Fat robots are said to move along, or to, *lanes*, which are the Minkowski sum of a line and the unit disk. That is, a lane is an infinite, straight path of unit thickness. The portion of a lane in which a fat robot moves in a single LCM cycle is its *path*.

### 7.4. Transparent Fat Robots in SSYNC

We now examine the first algorithm for fat robots, Algorithm $\{G, V, S, L\}$. Recall that fat robots are assumed to be initially grid-aligned in this chapter. Since the origin of the grid is located at $c$, there can be no robots initially located on the center line. Therefore, our fat robot algorithms operate completely independently on each half-plane without the need to append a special procedure as described for point robots in Section 6.9.

In Algorithms $\{G, V, S, \overline{L}\}$ and $\{G, V, S, \overline{L}\}$, the fat robots are transparent. This allows the fat robots to see the wall and the entire doorway. Therefore, all robots can determine the center of the doorway, $c$, as well. Thus, transparent fat robots can determine the directions up, down, left, and right at any time for themselves.

Many algorithmic details provided in this section will be used once more in Section 7.5.
7.4.1. Phase 1 - Form Unique Horizontal Lanes

The goal of the first phase is to place all of the robots within unique horizontal lanes. Recall that a lane is an infinite straight path of unit thickness. This provides each robot an infinite region in which to move without collisions.

In this phase, each robot will move strictly either leftwards or downwards in one LCM cycle in which it moves. In order to coordinate the movement of robots to avoid collisions, the robots agree on an ordering as follows: The robots order themselves from bottom to top with ties broken by the robot farther to the left coming first in the ordering. In order, robots are given clearance to move down if their path to their target location (which we will soon discuss) is empty and does not overlap with the path of any other robot granted clearance.

Since all robots see each other and they compute the same ordering, each robot can compute the chosen paths of all other robots.

Once as many robots as possible are granted clearance to move downwards, the robots who were not granted clearance yet are ordered from left to right, with ties being broken by the robot farther down coming first in the ordering. In order, robots are given clearance to move left if their path to their target location is empty and does not overlap with the path of any other robot granted clearance. Thus, in one LCM cycle, as many robots are possible are granted horizontal or vertical clearance to move left or down, respectively, along non-overlapping paths.

Before the robots move based on their computed clearances, the same method of computing clearances is repeated except they begin with the left-to-right ordering first.
Whichever of the two clearances allows the most robots to move is chosen by all robots. If equal, they all choose the prior clearance calculations.

Once the robots have determined their clearances, all activated robots with clearance to move will simultaneously move either left or down to their target locations. Their target locations depend on the number of robots in the current configuration that have moved out. A robot is classified as having already moved out if it satisfies the following conditions:

- It is in a unique horizontal lane, as are any robots below it.
- It is at least $n$ units below the door.
- There are at least $n$ units vertically between it and any robot not within a unique horizontal lane.

Once a robot has moved out, it will not recompute clearances or move again for the remainder of Phase 1.

If there are no robots yet that are classified as having moved out, then the right-most robot with vertical clearance moves downwards to a distance of $2n + s + 1$ below the bottom-most robot.

Subsequently, when there is at least one robot that has moved out, robots with vertical clearance move as follows: The $i^{th}$ robot with vertical clearance from right to left moves directly downwards to the $i^{th}$ empty horizontal lane from bottom to top that is above the bottom-most robot that has already moved out.
The movement of robots with horizontal clearance is similar. Specifically, the $i^{th}$ robot with horizontal clearance from bottom to top will move to the $i^{th}$ available vertical lane from right to left that is to the left of all robots with horizontal clearance. We say that a lane is available if it only contains robots that have already moved out.
7.4.2. Phase 2 - Form Unique Vertical Lanes

Phase 2 begins once all robots meet the criteria of having moved out. Thus, all of the robots occupy unique horizontal lanes. The goal of this phase is to move robots within their horizontal lanes to form unique vertical lanes as well.

The movement of Phase 2 is simple and requires only one epoch. Specifically, the \( i^{th} \) topmost robot moves within its lane to \( i - 0.5 \) from the wall. An example is provided in Figure 7.3.

![Figure 7.3. Movement of Robots in Phase 2](image.png)

7.4.3. Phase 3 - Move Robots Through the Door

Phase 2 begins once all robots are in both unique horizontal and vertical lanes. The goal of this phase is to move robots through the door in groups of \( s \) robots, i.e. as many as can simultaneously fit through the door.

From the staircase pattern formed in the previous phase, the rightmost \( s \) robots can move directly upwards simultaneously to be in front of the door without crossing paths. Specifically, the \( i^{th} \) robot from right to left moves vertically to \( i - 0.5 \) above the
edge of the doorway. An example where \( s = 2 \) is shown in Figure 7.4.

![Figure 7.4. Movement of Robots in Phase 3](image)

Once all \( s \) robots have moved to the height of the door, these robots are free to move to the door but do not yet disappear through it. Once all of the robots within the height of the door are at the door, the robots are free to disappear. Subsequently, the next group of robots moves into position in front of the door. The pattern continues until all robots have exited.

7.4.4. Algorithm \( \{G, V, S, L\} \)

The three phases of the algorithm are globally separated based on the conditions described in Algorithm 1. Then, the CA-pairs for Algorithm \( \{G, V, S, L\} \) are separated by phase for ease of comprehension.

7.4.4.1. Phase 1

In this phase, the robots move to form unique horizontal, but not necessarily vertical, lanes.
Algorithm 1: Phase Conditions

\[\begin{align*}
\text{if} \quad \text{robots occupy unique horizontal and vertical lanes or there is a robot in the doorway} & \quad \text{then} \\
& \quad \text{Phase 3} \\
\text{else if} \quad \text{robots occupy unique horizontal lanes} & \quad \text{then} \\
& \quad \text{Phase 2} \\
\text{else} & \quad \text{Phase 1}
\end{align*}\]

**Condition 1.1:** Robot \(r_i\) is the rightmost robot with vertical clearance, and no robots have “moved out”.

**Action 1.1:** Move vertically to a distance \(2n + s + 1\) below the bottom-most robot.

**Condition 1.2:** Robot \(r_i\) is the \(i^{th}\) robot from right to left with vertical clearance, and \(\geq 1\) robot has “moved out”.

**Action 1.2:** Move vertically to the \(i^{th}\) empty horizontal lane from bottom to top above the bottom-most robot that has already moved out.

**Condition 1.3:** Robot \(r_i\) is the \(i^{th}\) robot from bottom to top with horizontal clearance.

**Action 1.3:** Move horizontally to the \(i^{th}\) available vertical lane from right to left that is to the left of all robots with horizontal clearance.

7.4.4.2. Phase 2

In this phase, the robots move to form both unique horizontal and vertical, lanes.

**Condition 2.1:** Robot \(r_i\) is the \(i^{th}\) topmost robot.

**Action 2.1:** Move horizontally to \(i - 0.5\) from the wall.

7.4.4.3. Phase 3

In this phase, the robots move through the door.

**Condition 3.1:** Robot \(r_i\) is in the \(i^{th}\) lane from right to left where \(i \leq s\), and \(r_i\) is not
within the height of the door, and there are no robots in the doorway.

**Action 3.1:** Move vertically to \( i - 0.5 \) above the edge of the door.

**Condition 3.2:** Robot \( r_i \) is at the door, and there are no lanes within the height of the door containing a robot who isn’t at the door.

**Action 3.2:** Disappear through the door.

**Condition 3.3:** Robot \( r_i \) is in a unique horizontal lane within the height of the door, but not at the door, and all other lanes front of the door are occupied.

**Action 3.3:** Move rightwards to the door.

### 7.4.5. Analysis

**Lemma 7.1** No robot in Phase 1 collides with another robot in Phase 1.

**Proof:** Since the robots are transparent and operate in \( \text{SSYNC} \), each robot obtains the same snapshot, i.e. the position of every robot before the next move occurs. The robots are first ordered based on position, thus the ordering is agreed upon by all robots. Clearance to move is granted to robots in order, and a robot is only granted clearance if the robot has a clear path to its target position that does not overlap with another robot or the path of another robot during the next move. Thus, by design, robots move along strictly empty, non-overlapping paths in Phase 1 and therefore they cannot collide. \( \square \)

**Lemma 7.2** Robots move to unique horizontal lanes (Phase 1) in \( O(\sqrt{n}) \) epochs.

**Proof:** Consider the configuration of grid-aligned robots as a collection of columns and rows of robots existing within the SER, where the SER has sides aligned with the grid. From each side of the SER, begin from each cell along the side and continue inwards (per-
pendicularly to the side) until encountering a robot. The first robot encountered is called a \textit{free} robot, and the remaining robots are called \textit{blocked}. Suppose that there were $o(\sqrt{n})$ free robots corresponding to each side of the SER. Then, there is necessarily $o(n)$ robots inside of the SER, contradicting the assumption that there are $n$ robots initially. Therefore, if there are $k$ robots in the current configuration, then $\Omega(\sqrt{k})$ robots are free for a given configuration.

Suppose the free robots are removed from the configuration each step, allowing some previously blocked robots to become free. The number of robots who haven’t been free at step $t$ is given by the recurrence relation $R(0) = n$, $R(t) = R(t - 1) - \max(1, \lfloor \sqrt{R(t-1)} \rfloor)$. Therefore, all $n$ robots can become free and disappear in $t \leq 2\sqrt{n} = O(\sqrt{k})$ steps. Note that the same logic holds if robots are only freed on the left and bottom sides of the SER. Thus, $O(\sqrt{n})$ steps are sufficient for all $n$ robots to move from any initial configuration as long as the is no collision with robots that have previously moved, which we will show.

The free robots move out sufficiently far such that they are recognized by all robots as having \textit{moved out}, as described in Section 7.4.1. Further, they move such that robots that haven’t yet moved out can later move to their target positions without collisions. Specifically, consider the two cases of free robot movement below:

1. Robots moving horizontally (left): Let $\mathcal{F}_L(t)$ represent the set of free robots at a step $t$ moving left. By virtue of being a free robot on the left side of the SER, each robot has an unobstructed half-lane extending infinitely far to the left. The bottom-most robot in $\mathcal{F}_L(t)$ will move to the rightmost available vertical lane to the left all robots in $\mathcal{F}_L(t)$. The second bottom-most robot in $\mathcal{F}_L(t)$ moves to the next nearest available vertical lane, and so on. All of the robots that move horizontally in this step are then free to move vertically via non-overlapping paths in the next step. If not all robots with clearance are activated, $\Omega(\sqrt{k})$ robots will have clearance to move in the subsequent step using the previous logic, where $k$ is the
number of robots that haven’t moved out.

2. Robots moving vertically (down): Let $\mathcal{F}_D(t)$ represent the set of free robots at a step $t$ moving down. By virtue of being a free robot on the bottom side of the SER, each robot has an unobstructed half-lane extending infinitely far down or until reaching a robot that has previously moved out. The robots can move along parallel lanes to unique horizontal lanes while moving sufficiently far to leave enough available lanes for subsequent robots while not moving far enough to collide with a previously moved out robot. Specifically, if no robots satisfy the conditions for having moved out, the rightmost robot in $\mathcal{F}_D(t)$ will move to $2n + s + 1$ units below the bottom-most robot. The second rightmost will move to a horizontal lane one unit higher, and so on. If there has been a robot that has already moved out, then the mapping of robots in $\mathcal{F}_D(t)$ to horizontal lanes is as before where the rightmost robot goes to the bottom-most available lane above the bottom-most robot that has moved out.

If not all robots with clearance are activated, then their locations are still available to them when they are subsequently activated are still available because otherwise would imply that they did not have clearance in the previous step.

There are no gaps in the horizontal lanes of robots that have already moved out because:

1. If all robots with vertical clearance are active and therefore move, then they move along non-overlapping paths to consecutive lanes.

2. If only a subset of the robots with vertical clearance are activated, this may leave one or more available horizontal lane between lanes occupied by robots that have moved out. However, the robots that have moved out will not move again until all robots have moved out. Therefore, the vertical lanes the previously inactive robots would have taken to reach the available lane are still clear, and they are free to move to that lane in a subsequent step, unless the robot determines another robot will move to that lane based on the ordering or robots.

Therefore, once each robot has moved out, all robots are below the height of the door and in unique horizontal lines, satisfying the conditions for completing Phase 1.

As previously shown, at each step, $\Omega(\sqrt{k})$ robots can perform one of the $\leq 2$ moves needed to reach a position to complete Phase 1. If a robot is not activated in a given step, then it will still be free to be activated in the subsequent step, i.e. within one
epoch. Thus, robots move to unique horizontal lanes below the door completing Phase 1 in $O(\sqrt{n})$ epochs.

**Lemma 7.3** No robot in Phase 2 collides with another robot in Phases 1 or 2.

**Proof:** Once in Phase 2, the robots all lie within non-overlapping horizontal lanes. The robots move strictly within these lanes during Phase 2. Thus, no robot in Phase 2 collides with another robot in Phase 2. A robot in Phase 2 cannot collide with a robot in Phase 1 because Phases 1 and 2 are globally separated across $SSYNC$ LCM cycles based on the condition of all robots occupying unique horizontal lanes, therefore no robot is in Phase 1 while another is in Phase 2.

**Lemma 7.4** Once Phase 1 is complete, robots complete Phase 2 in $O(1)$ epochs.

**Proof:** Phase 2 only requires each robot to move exactly once, and a robot knows its target location based on the ordering of lanes from top to bottom, i.e. the rank of its lane. Therefore, a robot can move to its target position without waiting once activated in Phase 2. Thus, all robots move to their final positions of Phase 2 in 1 epoch.

**Lemma 7.5** No robot in Phase 3 collides with another robot in Phases 1, 2, or 3.

**Proof:** Similarly to Phase 2, there is no overlap between Phase 3 and prior phases. This is because all robots must satisfy the condition of Phase 3 requiring all robots to lie within distinct vertical and horizontal lanes below the height of the door, and thereafter the conditions are maintained. Specifically, robots must first move upwards within their current, non-overlapping vertical lanes to unique horizontal lanes in front of the door (Step 1). Therefore, there cannot be collisions between robots executing this movement and all robots still satisfy the conditions of being in Phase 3. Then, the robots in front of the door take their unique horizontal lanes to the door (Step 2). After these robots disappear
(Step 3), the process begins again with the remaining robots still satisfying the conditions for Phase 3. Note that multiple robots in the doorway may not be in unique vertical lanes, but robots in the doorway satisfy the other possible condition for Phase 3. Therefore, no two robots executing Phase 3 Step 2 or 3 collide and no robot in Phase 3 can collide with a robot in a previous phase.

Suppose a subset of the robots performing Step 1 of Phase 3 move up. Then, the robots within the height of the door will remain in place until the horizontal lanes in front of the door are filled. Only then may they all move along non-overlapping horizontal paths to the door. Therefore, a robot executing Step 1 of Phase 3 cannot collide with a robot executing Step 2.

Lemma 7.6 Once Phase 2 is complete, robots complete Phase 3 in $O(n/s)$ epochs.

Proof: Suppose that the robots are operating in $\mathcal{FSYNC}$, then $s$ robots can move upwards simultaneously in 1 LCM cycle, then move to the door in another cycle, and finally disappear. Subsequently, the next group of $s$ robots moves up and the process repeats. Since there are $n$ total robots, this process repeats $\lceil n/s \rceil$ times. Thus, in $\mathcal{FSYNC}$, Phase 3 runs in $O(n/s)$ steps.

Now consider $\mathcal{SSYNC}$. Suppose a subset of the robots performing Step 1 of Phase 3 move up. In this case, the robots within the height of the door will remain in place until the horizontal lanes in front of the door are filled, which occurs within one epoch. Subsequently robots will move through to the door but not disappear, again in one epoch. Then, once all of the robots that were in front of the door are in the doorway, they are free to disappear. All will be activated again and disappear then within one epoch. Subsequently, the next group of $s$ robots move and the process repeats. Therefore, in $\mathcal{SSYNC}$,
Phase 3 runs in \( O\left(\frac{n}{s}\right) \) epochs.

\[ \square \]

**Theorem 7.7** Algorithm \( \{G, V, S, L\} \) solves Doorway Egress in \( O(\sqrt{n} + \frac{n}{s}) \) time without collisions.

\[ \square \]

### 7.5. Transparent Fat Robots with Lights

This section considers Algorithm \( \{G, V, S, L\} \), i.e., (initially grid-aligned) transparent, fat robots with lights operating in ASYNC. We begin by describing the phases of the algorithm.

#### 7.5.1. Phase 1 - Form Unique Horizontal Lanes

Once again, goal of the first phase is to position all of the robots within unique horizontal lanes.

In this phase, each robot can only move leftwards or downwards in a single move. Robots begin colored *Initial*. As in Algorithm \( \{G, V, S, L\} \), the robots begin by computing their clearances. However, since this algorithm runs in ASYNC, the robots do not move until all robots are ready; this ensures that all robots use the same snapshot to compute clearances. Once a robot knows whether it has clearance to move vertically or horizontally or if it has no clearance at all, it colors itself *Clear\(_v\)*, *Clear\(_h\)*, *Blocked*, respectively.

When there are no robots that remain colored *Initial*, the robots colored *Clear\(_v\)* behave in much the same way as Algorithm \( \{G, V, S, L\} \). Specifically, if there are no robots yet colored *Out*, then the rightmost robot colored *Clear\(_v\)* moves downwards to a distance of \( 2n + s + 1 \) below the bottom-most robot and changes its colored to *Out*. Then, when there is at least one robot colored *Out*, the \( i^{th} \) robot colored *Clear\(_v\)* from right to
left moves directly downwards to the $i^{th}$ empty horizontal lane from bottom to top that is above the bottom-most robot colored Out, then it changes its color to Out as well.

The robots with horizontal clearance act similarly to the robots executing Algorithm $\{G, V, S, L\}$ as well. If there are no robots yet colored Align, then the bottom-most robot colored Clear$_h$ moves leftwards to a distance one unit farther left than the leftmost robot and changes its colored to Align. Then, when there is at least one robot colored Align, the $i^{th}$ robot colored Clear$_h$ from bottom to top moves directly leftwards to the $i^{th}$ vertical lane that is not occupied by a robot starting from the vertical lane containing the robot colored Align and continuing leftwards.
We use the bottom-most robot to check whether all of the robots have finished moving in accordance with their clearances. Specifically, the bottom-most robot colored \texttt{Out} checks if all robots are colored one of: \texttt{Clear}_v, \texttt{Clear}_h, \texttt{Blocked}, \texttt{Align}, or \texttt{Out}, and it checks if the $i^{th}$ robot colored \texttt{Clear}_h from bottom to top is in an empty vertical lane that is $i$ units to the left of the robot colored \texttt{Align} for all robots colored \texttt{Clear}_h. Once these conditions are satisfied, the bottom-most robot colored \texttt{Out} changes its color to \texttt{Wait}.

When any robot other than one colored \texttt{Out} or \texttt{Wait} sees a robot colored \texttt{Wait}, it recomputes its clearances and changes its color accordingly. Once the robot colored \texttt{Wait} sees that the colors of each robot are consistent with clearances it computed, it changes its color back to \texttt{Out}.

This process continues until all robots are eventually colored \texttt{Out}, and occupy unique horizontal lanes.
7.5.2. Phase 2 - Form Unique Vertical Lanes

Phase 2 begins when all robots are colored Out. From there, robots move horizontally to $n + 2 - i$ units from the wall, where $r_i$ is the $i^{th}$ robot from bottom to top, then change color to Stair. Once all robots have moved and changed their color, they occupy unique horizontal and vertical lanes. This phase occurs in a single epoch.

![Figure 7.7. Movement of Robots in Phase 2](Image)

7.5.3. Phase 3 - Move Robots Through the Door

Phase 3 begins once all robots are colored Stair and are in both unique horizontal and vertical lanes. The goal of this phase is to move robots through the door in groups of $s$ robots, i.e. as many as can simultaneously fit through the door.

When a robot wakes up colored Stair, is one of the $s$ rightmost robots, and there are no robots within the height of the door, it changes its color to Terminal. Once a robot is colored Terminal and not within the height of the door, and there are $s$ robots colored terminal, it can move vertically to $i - 0.5$ above the edge of the doorway, where $r_i$ is the $i^{th}$ robot from right to left.
Once all \( s \) robots colored \texttt{Terminal} have moved to their target positions within the height of the door, these robots are free to move to the door and disappear through it. Subsequently, the next group of robots moves into position in front of the door. The pattern continues until all robots have exited.

7.5.4. Algorithm \{\texttt{G, V, S, L}\}

The algorithm is provided in the form of condition-action pairs. Since there is significant overlap between conditions across colors (e.g. Condition 1), we do not separate CA-pairs by color.

\underline{Condition 1}: Robot \( r_i \) is colored \texttt{Initial}.

\textbf{Action 1}: Change color to \texttt{Clear\_v}, \texttt{Clear\_h}, or \texttt{Blocked} in accordance with computed clearances.

\underline{Condition 2}: There is a robot colored \texttt{Wait}, and robot \( r_i \) is not colored \texttt{Out} or \texttt{Wait}.

\textbf{Action 2}: Change color to \texttt{Clear\_v}, \texttt{Clear\_h}, or \texttt{Blocked} in accordance with computed clearances.

Figure 7.8. Movement of Robots in Phase 3
clearances.

**Condition 3:** Robot $r_i$ is the rightmost robot colored \texttt{Clear$_v$} and there is no robot colored \texttt{Out} or \texttt{Initial}.

**Action 3:** Move directly downwards to a distance $2n+s+1$ below the height of the bottommost robot then change color to \texttt{Out}.

**Condition 4:** Robot $r_i$ is the $i^{th}$ robot colored \texttt{Clear$_v$} from right to left, there is a robot colored \texttt{Out}, and there is no robot colored \texttt{Initial}.

**Action 4:** Move directly downwards to the $i^{th}$ bottom-most horizontal lane that is not occupied by a robot colored \texttt{Out} and is above the height of the bottom-most robot colored out, then change color to \texttt{Out}.

**Condition 5:** Robot $r_i$ is the bottom-most robot colored \texttt{Clear$_h$} and there is no robot colored \texttt{Align}.

**Action 5:** Move directly leftwards to a distance one unit farther left than the leftmost robot, then change color to \texttt{Align}.

**Action 6:** Robot $r_i$ is the $i^{th}$ robot colored \texttt{Clear$_h$} from bottom to top, there is a robot colored \texttt{Align}, and there is no robot colored \texttt{Initial}.

**Action 6:** Move directly leftwards to the $i^{th}$ vertical lane that is not occupied by a robot starting from the lane containing the robot colored \texttt{Align} and continuing leftwards.

**Condition 7:** Robot $r_i$ is colored \texttt{Out} and there is a robot colored \texttt{Stair}, or all robots are colored \texttt{Out}.

146
Action 7: Move horizontally to $n + 2 - i$ units from the wall, where $r_i$ is the $i^{th}$ robot from bottom to top, then change color to Stair.

Condition 8: Robot $r_i$ is the bottom-most robot colored Out, all robots are colored one of: Clear_v, Clear_h, Blocked, Align, or Out, and the $i^{th}$ robot colored Clear_h from bottom to top is in an empty vertical lane that is $i$ units to the left of the robot colored Align for all robots colored Clear_h.

Action 8: Change color to Wait.

Condition 9: Robot $r_i$ is colored Wait and the color of all other robots is consistent with the clearances $r_i$ computed.

Action 9: Change color to Out.

Condition 10: Robot $r_i$ is colored Stair and is one of the $s$ rightmost robots, and there are no robots within the height of the door.

Action 10: Change color to Terminal.

Condition 11: Robot $r_i$ is colored Terminal, and all robots colored Terminal are in unique horizontal lanes within the height of the door as described in Action 12.

Action 11: Move horizontally through the door and disappear.

Condition 12: Robot $r_i$ is colored Terminal, there are $s$ robots colored Terminal, and $r_i$ is not within the height of the door.

Action 12: Move vertically to $i - 0.5$ units above the edge of the door, where $r_i$ is the $i^{th}$ robot from right to left.
Lemma 7.8 No robot in Phase 1 collides with another robot in Phase 1.

Proof: The robots all have visibility of each other, and every robot determines where each robot is going at each stage. In order, robots are given clearance to move if they have a clear path to their target, and a robot that comes later in the ordering will not move if its path would cross with another robot whose path has already been determined. Therefore, these robots move strictly along empty, non-overlapping paths. Thus, they cannot collide. □

Lemma 7.9 Robots move to unique horizontal lanes (Phase 1) in $O(\sqrt{n})$ epochs.

Proof: The goal of Phase 1 is to move all of the robots out of the initial configuration to unique horizontal lanes and color the robots Out in $O(\sqrt{n})$ epochs. Robots begin all colored Initial. As in Algorithm $\{G, V, S, L\}$, all robots determine which robots have horizontal or vertical clearance. These robots all change their color accordingly in one epoch. No robot moves until all robots are colored either Clear_v, Clear_h, or Blocked. That is to say, robots will not move while there is a robot colored Initial.

Once all robots have lights consistent with their clearances, there are two ways robots can move in Phase 1:

1. Robots moving left (colored Clear_h): Once there are no robots colored Initial, the bottom-most robot colored Clear_h moves leftwards in its horizontal lane to 1 unit farther left than the leftmost robot without waiting in one epoch. At the end of the movement, the bottom-most robot colored Clear_h changes its color to Align. All other robots colored Clear_h will not move until this occurs. Then, the remaining robots colored Clear_h have a consistent starting point from which to do the following: The $i^{th}$ robot from bottom to top colored Clear_h moves within its horizontal lane to $i$ units farther leftwards than the robot colored Align. As the robots move, the position of the robot colored Align does not change, and neither does their ranking from top to bottom. Therefore, the remaining robots
colored \texttt{Clear}.\texttt{h} can all move to their target positions in one epoch without concern for the movement of other robots. Thus, robots colored \texttt{Clear}.\texttt{h} all move to unique vertical lanes in $O(1)$ epochs.

2. Robots moving down (colored \texttt{Clear}.\texttt{v}): Once there are no robots colored \texttt{Initial}, the rightmost robot colored \texttt{Clear}.\texttt{v} moves downwards in its vertical lane to $2n + s + 1$ units below the bottom-most robot without waiting in one epoch. At the end of the movement, the rightmost robot colored \texttt{Clear}.\texttt{v} changes its color to \texttt{Out}. All other robots colored \texttt{Clear}.\texttt{v} will not move until this occurs. Then, the remaining robots colored \texttt{Clear}.\texttt{v} have a consistent starting point from which to do the following: The $i^{th}$ robot from right to left colored \texttt{Clear}.\texttt{v} moves within its vertical lane to $i$ units above the bottom-most robot colored \texttt{Out}, then it too changes its color to \texttt{Out}. As the robots move, the position of the bottom-most robot colored \texttt{Out} does not change, and neither does their ranking from right to left. Therefore, the remaining robots colored \texttt{Clear}.\texttt{v} can all move to their target positions in one epoch without concern for the movement of other robots colored \texttt{Clear}.\texttt{v}. Thus, robots colored \texttt{Clear}.\texttt{v} all move to unique horizontal lanes in $O(1)$ epochs.

Once all robots colored \texttt{Clear}.\texttt{v} or \texttt{Clear}.\texttt{h} have moved to their target positions, all robots are colored \texttt{Align}, \texttt{Clear}.\texttt{h}, \texttt{Blocked}, or \texttt{Out}. When only robots with the aforementioned colors are visible and the robots colored \texttt{Clear}.\texttt{h} are in their previously described target position relative to the robot colored \texttt{Align}, the robots begin the next \textit{mega-cycle}, i.e. a single iteration of robots moving once in accordance with their clearances. To do this, the bottom-most robot (colored \texttt{Out}) changes its color to \texttt{Wait}. The other robots colored \texttt{Out} do not change their color. While there is a robot colored \texttt{Wait}, no robots move, and all robots colored not colored either \texttt{Out} or \texttt{Wait} will recompute clearances and change their color to either \texttt{Clear}.\texttt{h}, \texttt{Clear}.\texttt{v}, or \texttt{Blocked} accordingly.

Since all robots obtain the same snapshot of the configuration and execute the same algorithm, the robot colored \texttt{Wait} can determine when the colors of all robots are consistent with the clearances it determined. When this the colors of robots match the configuration as previously described, the robot colored \texttt{Wait} changes its color back to \texttt{Out}. Therefore, within 3 epochs of robots moving to their target positions based on their clearance in this
stage, the next mega-cycle begins.

Thus, a single mega-cycle of robots moving in accordance with their calculated clearances requires $O(1)$ epochs.

As in Algorithm \{G, V, S, L\}, $O(\sqrt{n})$ steps of robots moving in accordance with their clearances are sufficient for all robots to achieve unique horizontal lanes. Thus, in $O(\sqrt{n})$ epochs, all robots are in unique horizontal lanes and are colored $\texttt{Out}$. This completes Phase 1.

**Lemma 7.10** No robot in Phase 2 collides with another robot in Phases 1 or 2.

**Proof:** Once in Phase 2, the robots robots all lie within non-overlapping horizontal lanes. The robots move strictly within these lanes during Phase 2. Thus, no robot in Phase 2 collides with another robot in Phase 2. A robot in Phase 2 cannot collide with a robot in Phase 1 because Phase 2 cannot begin until all robots have changed their color to $\texttt{Out}$, and therefore there is no overlap between phases.

**Lemma 7.11** Once Phase 1 is complete, robots complete Phase 2 in $O(1)$ epochs.

**Proof:** Once every robot is colored $\texttt{Out}$ (and is in a unique horizontal lane) or there is a robot colored $\texttt{Stair}$, each robot can then move to $n + 2 - i$ units from the wall, where $r_i$ is the $i^{th}$ robot from bottom to top, and change its color to $\texttt{Stair}$ without waiting in 1 epoch. Note that the ranking of robots from bottom to top is invariant in Phase 2, and therefore so are their target positions. Therefore, all robots move to their target position of Phase 2 regardless of the movement of others in 1 epoch. Thus, all robots move to their final positions of Phase 2 and change their color to $\texttt{Stair}$ in $O(1)$ epochs.

**Lemma 7.12** No robot in Phase 3 collides with another robot in Phases 1, 2, or 3.

**Proof:** Similarly to Phase 2, there is no overlap between Phase 3 and prior phases. This
is because all robots must satisfy the condition of Phase 3 requiring all robots to be colored either Stair or Terminal, and thereafter the conditions are maintained. Thus, no robot in Phase 3 collides with a robot from a prior phase. In Phase 3, robots must first move upwards within their current, non-overlapping vertical lanes to unique horizontal lanes in front of the door (Step 1). Therefore, there cannot be collisions between robots executing this movement and all robots still satisfy the conditions of being in Phase 3. Then, the robots in front of the door take their unique horizontal lanes to the door and disappear (Step 2). After these robots disappear, the process begins again with the remaining robots still satisfying the conditions for Phase 3. Therefore, no two robots executing Phase 3 Step 2 collide.

Suppose a subset of the robots performing Step 1 of Phase 3 move up. Then, the robots within the height of the door will remain in place until the horizontal lanes in front of the door are filled. Only then may they all move along non-overlapping horizontal paths to the door. Therefore, a robot executing Step 1 of Phase 3 cannot collide with a robot executing Step 2.

Lemma 7.13 Once Phase 2 is complete, robots complete Phase 3 in $O\left(\frac{n}{s}\right)$ epochs.

Proof. Suppose that the robots are operating in $\mathcal{FSYNC}$, then $s$ robots change their color to Terminal, performing Action 10, in 1 LCM cycle. Then these $s$ robots colored Terminal will perform Action 12 and move upwards upwards simultaneously in 1 LCM cycle. Finally, these robots can perform Action 11, i.e. move horizontally through the door and disappear in 1 LCM cycle. Subsequently, the next group of $s$ robots moves up and the process repeats. Since there are $n$ total robots, this process repeats $\left\lceil \frac{n}{s} \right\rceil$ times. Thus, in $\mathcal{FSYNC}$, Phase 3 runs in $O\left(\frac{n}{s}\right)$ steps.
Now consider \textit{ASYNC}. At the beginning of Phase 3, all robots are colored \texttt{Stair}. Subsequently the rightmost \( s \) robots satisfy Condition 10, and change their color to \texttt{Terminal} (Step 1). Robots will not satisfy Conditions 11 or 12 until all \( s \) robots have performed Action 10 in 1 epoch. Subsequently, these robots satisfy Condition 12 but not 10 or 11. Thus, they move to unique horizontal lanes within the height of the door (Step 2). Their target locations are fixed since their rank from right to left is fixed as they move up. Robots will not satisfy Condition 11 until all robots have moved into position in front of the door, which occurs in 1 epoch. Finally, once all \( s \) robots have reached their known target positions, they satisfy Condition 11 but not 10 or 12. The robots are then free to move through the door and disappear (Step 3), which occurs in 1 epoch. Note that even as robots disappear, the robots that have yet to move horizontally through the door still satisfy Condition 11 but not 10 or 12. Until all of the robots colored \texttt{Terminal} have disappeared, none of the Conditions 10-12 are satisfied by the next group of \( s \) robots, or any others. Once they have all disappeared, however, only the rightmost \( s \) robots satisfy Condition 10 and the pattern continues. Notice that there is no overlap overlap between steps and the steps are unaffected by asynchrony. Therefore, each group of \( s \) robots completes Phase 3 and disappears in \( O(1) \) epochs. Thus, in \textit{ASYNC}, Phase 3 runs in \( O \left( \frac{n}{s} \right) \) epochs.

\textbf{Theorem 7.14} Algorithm \( \{G, V, \overline{S}, L\} \) solves \textsc{Doorway Egress} in \( O \left( \sqrt{n} + \frac{n}{s} \right) \) epochs without collisions using 9 colors.
7.6. Conclusion

In this chapter, we have examined the problem of Doorway Egress for fat robots and provided two optimal (in terms of time) algorithms. If distance and space are prioritized over time, the linear-time algorithm can be chosen as it is optimal in the other performance measures. It would be interesting to try to reduce the distance and spatial complexities of the time-optimal algorithms so that there is no need to prioritize either time or energy-efficiency and space. It would also be of interest to remove the initial assumption of grid-alignment, and to provide algorithms for opaque, fat robots.
Chapter 8. Doorway Egress Lower Bound Conjecture

In this chapter, we look at Doorway Egress once again in consideration of the lower bound on time for robots with no additional capabilities. That is, we consider \( \{V, S, L\} \) for point robots and \( \{G, V, S, L\} \) for fat robots, granting that the fat robots may be initially grid-aligned. We would like to show that, without lights, visibility, or some level of synchrony, \( \Omega(n) \) time is required. This would make our linear-time algorithm for either point or fat robots optimal in terms of time, distance, and space. Further, it would show that robots require at least one ability in order to achieve sublinear time, which would imply that our algorithms cannot be improved upon by removing abilities without increasing the upper bounds on time. We have made progress towards this end but have not fully established this as a result. Therefore, this chapter describes our conjecture and the techniques that can be employed by others in future examination of this problem.

For simplicity, we will refer to Doorway Egress in the \( \{V, S, L\} \) or \( \{G, V, S, L\} \) model as Weak Doorway Egress (WDE) for point or fat robots, respectively. Let us start by considering fat robots.

8.1. 1-WDE Algorithms

In this section, without further mention, we consider algorithms for WDE in which robots movement only once. That is, each robot must move directly from its initial position to the door. We call such algorithms 1-WDE.

**Lemma 8.1** For any 1-WDE algorithm, \( \Omega(n) \) time is required for \( n \) fat, opaque robots in any collinear configuration to move directly to the door under any scheduler.
Proof. Suppose that there are \( n \) collinear robots in a configuration of robots performing WDE, but these robots may only move directly to the door. That is, robots cannot move at all before their final movement.

Suppose, for now, that the robots are vertically-aligned, i.e., parallel to the door. Notice that with obstructed visibility and obliviousness, a robot can only determine if it is the topmost robot, bottom-most robot, or an interior robot.

Let the activation schedule happen to be fully-synchronous. Suppose only the topmost and/or bottom-most robots move each round. Then only \( \Theta(1) \) robots move to the door each round, requiring \( \Theta(n) \) rounds for all robots to move off the line to the door, and we are done.

Otherwise, a subset of the interior robots on the vertical line moves to the door. Recall that these robots are opaque. Therefore, each robot can only see its neighbors on the vertical line. If a terminal robot \( r_i \) that is higher (resp., lower) on the line than another terminal robot, \( r_j \), moves to a lower (resp., higher) position on the door than \( r_j \), their paths necessarily cross and thus an execution exists in which a collision occurs.

Thus, the \( i^{th} \) terminal robot (moving to the door in the current round) from top-to-bottom must move to the \( i^{th} \) chosen target position on the door from top-to-bottom in order to avoid collisions in every round. For any set of robots, we say the \( i^{th} \) robot from top to bottom has rank \( i \).

Consider a round in which \( k = \omega(1) \) robots move from the vertical line to the door. If there is no such round, then \( \Theta(n) \) rounds are required, and we are done. In the line of robots \( \{r_1, \ldots, r_k\} \), a robot with rank \( 2 \) through \( k - 1 \) cannot determine its rank as we’ve previously noted; otherwise would imply the ability to see beyond adjacent robots.
Now suppose that there are $k$ terminal robots that maintain rank (i.e., maintain the same ordering from top to bottom in any vertical line passing through each robot’s path to the door) as they move to the door without all explicitly determining their rank. This implies that any higher (resp., lower) position on the line of robots is mapped to a higher (resp., lower) position on the door. If all robots are initially at least $n$ units below the door, there exists a configuration in which any robot can receive the same snapshot from any position and rank $\{r_2, \cdots, r_{k-1}\}$. Otherwise, for $n >> s$, grant that the robots in front of the door move through and the remaining robots on the line are all below the doorway.

Suppose, for the purpose of contradiction, that there is an $o(n)$ time, collision-free algorithm, $\mathcal{A}$. Then consider a snapshot that would cause a robot to move to a target position with $\Theta(1)$ positions on the doorway below it. If there is no such snapshot, then only $\Theta(1)$ target positions on the door are used every round, contradicting the assumption that the algorithm runs in $o(n)$ time. If there is such a snapshot, let the robot that receives the snapshot be $r_2$, thus only $\Theta(1)$ on the line below it can move to the door without causing a collision, and $r_1$ is the only robot that can move into the portion of the doorway above $r_2$’s target position without crossing $r_2$’s path. However, since $\mathcal{A}$ is deterministic, either $\omega(1)$ terminal robots below $r_2$ move to the door and a collision occurs for the described configuration, or $\{r_4, \cdots, r_{k-1}\}$ would never move to the door regardless of whether $r_2$ was there (in the current round or subsequent rounds), contradicting the assumption that $\omega(1)$ robots move to the door. In either case, this either contradicts the assumption of a collision-free or $o(n)$ time algorithm. Thus, either the algorithm is not collision-free or $\Theta(n)$ time is required.
Since $\Omega(n)$ time is required to avoid collisions in FSYNC, the logic immediately extends SSYNC and ASYNC as well.

Now consider if the configuration is still collinear, but is not a vertical line. E.g., consider if the line of robots were horizontal or diagonal. Consider any vertical slice through the robots’ paths to the door. If path $P_i$ is above $P_j$ in slice $S_m$, then there is no slice $S_n$ in which $P_i$ is below $P_j$, otherwise the paths necessarily cross and an execution exists in which a collision occurs. The prior logic holds for any collinear configuration. □

**Lemma 8.2** For any algorithm, $\Omega(m)$ time is required for $\Theta(m)$ fat, opaque robots in any collinear configuration to move directly to the door.

**Proof:** Now suppose that $m < n$ robots are in a collinear initial configuration and must move directly to the door as in Lemma 8.1, but $n - m$ robots may move multiple times and to any position.

Call the $n - m$ non-collinear robots *guiding robots*. We know from Lemma 8.1 that, without any guiding robots, linear time is required.

Now, if every configuration of these guiding robots led to the robots on the line maintaining rank as they move to the door, then the guiding robots are not needed since rank is maintained regardless (independently) of the positions of the guiding robots. Therefore, let us consider two types of positions for the guiding robots, *correct-guidance* and *incorrect-guidance*. In correct-guidance positions, the guiding robots are in positions such that the terminal robots on the line move to the door while maintaining rank. If the guiding robots are in incorrect-guidance positions, at least two terminal robots do not maintain rank and therefore collide.

Let the guide robots be initially placed adversarially, i.e., in incorrect-guidance po-
sitions. With obliviousness and no lights, the robots have no ability to distinguish between the first epoch or any subsequent epoch. They also have no means of direct communication. Therefore, position alone is all that can be used by the guiding robots to convey information. Since the algorithm is deterministic, and there is no ability to distinguish between rounds, we can set up the initial configuration such that the robots on the line move in accordance with the incorrect-guidance positions, resulting in a collision.

8.2. WDE Algorithms

In this section, we return to the more general WDE problem. Specifically, we describe an activation schedule that attempts to reproduce the challenges of $1$–WDE algorithms in any WDE algorithm.

**Lemma 8.3** For any algorithm, there exists an adversarial schedule in ASYNC such that, in each robot $r_i$’s snapshot, every active terminal robot is collinear with $r_i$.

**Proof:** Consider an asynchronous activation schedule such that robots are activated in order of decreasing perpendicular distance from the wall (left-to-right). In this schedule, if a robot performs a terminal movement, its path to the door must cross the vertical line passing through every robot that has yet to be activated. Let the robot speeds be such that when each robot $r_i$ performs its “look”, the terminal robots that have been activated earlier in the epoch are vertically aligned with $r_i$. Call this line the *sweep line*. Thus, when any robot $r_i$ is activated, it and all active terminal robots are on the sweep line.

We’ve shown that robots moving from a single line directly to the door takes linear time, even if there are non-terminal robots that can act as guides. We’ve then shown that there exists an activation schedule in which every active terminal robot is collinear.

158
at any given time. The next step would be to show that, using the adversarial, sweeping activation schedule, any algorithm for WDE by fat robots would require $\Omega(n)$ time. However, one issue with using the previous lemmas to that end is that the robots are not initially collinear. That is, the line of collinear terminal robots is progressively constructed. Perhaps the robots can perform several set-up steps such that they begin a terminal movement with some guarantees about the number of robots above or below them. Our algorithms make use of a similar method, where the robots first perform a series of steps before they all travel perpendicularly to the door during their terminal movements.

The problem is more challenging for point robots, where all $n$ robots can move themselves to the door simultaneously. Therefore, while the robots must still maintain rank while moving to the door, no robot can limit the number of target positions above or below it on the door based on its chosen target position. It is left as future work to expand on the prior lemmas and show that our algorithm for either point or fat robots with no additional abilities is time-optimal. Thus, we make the following conjecture.

**Conjecture 8.4** Any algorithm for WDE by either point or fat robots requires $\Omega(n)$ time.
Chapter 9. Arbitrary Pattern Formation on a Grid

In this chapter, we study the Arbitrary Pattern Formation problem in the classical model, i.e., transparent, point robots without lights. Let the robots be in any initial configuration in which they are located at distinct points. The problem is to relocate the robots such that each robot is at a distinct point forming an arbitrary geometric pattern given as input. Arbitrary Pattern Formation is fundamental in the context of swarm robotics.

We consider Arbitrary Pattern Formation on the integer plane, or grid, which restricts each move of a robot from its current position (grid point) to one of the four neighboring grid points. We measure the efficiency of our Arbitrary Pattern Formation algorithm through two measures, namely,

- *Time*: How long does it take robots to relocate to a configuration that matches the given target pattern and remain stationary thereafter? Time is measured using a standard notion of epochs, where an epoch is a duration in which each robot completes at least one LCM cycle.

- *Moves*: How many times a robot needs to move before reaching the pattern (target) position? The number of moves is measured in units of length on the grid.

The number of moves is a performance metric that has been considered for Arbitrary Pattern Formation on a grid by Bose et al. [45], where the authors provided both lower and upper bounds. The number of moves metric becomes important when robots are energy-constrained and need to minimize the number of times they move. The time metric becomes important when robots need to relocate to different target patterns frequently.
Table 9.1. Results on ARBITRARY PATTERN FORMATION for \( n \geq 1 \) robots on a grid in the asynchronous setting

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model</th>
<th>Time (in epochs)</th>
<th>Number of moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Any</td>
<td>( \Omega(\max{D^i, D^p}) ) (Th. 9.1)</td>
<td>( \Omega(\max{D^i, D^p}) ) [45]</td>
</tr>
<tr>
<td>Bose et al. [45]</td>
<td>Classical</td>
<td>( - )</td>
<td>( O((\max{D^i, D^p})^2) )</td>
</tr>
<tr>
<td>Our Algorithm</td>
<td>Classical</td>
<td>( O(\max{D^i, D^p}) )</td>
<td>( O(\max{D^i, D^p}) )</td>
</tr>
</tbody>
</table>

9.1. Contributions of This Chapter

In this chapter, we establish two results (summarized in Table 9.1). Result 1 is a lower bound on time and Result 2 is an asynchronous algorithm for ARBITRARY PATTERN FORMATION in the classical model. Let \( C_{\text{init}} \) be the initial configuration of \( n \geq 1 \) robots positioned on \( n \) distinct points (vertices) of a grid. Let \( C_{\text{pattern}} \) be the target pattern configuration of \( n \) robots positioned also on \( n \) distinct points. Let \( D^i \) and \( D^p \) be the diameters of \( C_{\text{init}} \) and \( C_{\text{pattern}} \), respectively, i.e., the maximum distance between any two robots (resp., target positions) in \( C_{\text{init}} \) (resp., \( C_{\text{pattern}} \)).

- **Result 1**: A time lower bound of \( \Omega(\max\{D^i, D^p\}) \) epochs for any (deterministic or randomized) ARBITRARY PATTERN FORMATION algorithm on a grid in the classical and/or the lights model, provided that no scaling is allowed on the pattern to be formed.

- **Result 2**: An asynchronous ARBITRARY PATTERN FORMATION algorithm with time \( O(\max\{D^i, D^p\}) \) and number of moves \( O(\max\{D^i, D^p\}) \) in the classical model.

Bose et al. [45] provided a lower bound of \( \Omega(\max\{D^i, D^p\}) \) on the number of moves for a robot in the setting of Result 1 (with no scaling of the pattern). Given these time and number-of-moves lower bounds, Result 2 has (asymptotically) optimal time and number-of-moves complexities. Furthermore, Bose et al. [45] proved an upper bound of \( O((\max\{D^i, D^p\})^2) \) on the number of moves for a robot in their ARBITRARY PATTERN
Formation algorithm. Given their result, Result 2 improves the number of moves bound by a factor of $\max\{D^i, D^p\}$. Additionally, Bose et al. [45] considered only asymmetric $C_{\text{init}}$ (hence a deterministic algorithm), whereas Result 2 works for any $C_{\text{init}}$ (using randomization).

9.2. Previous Work

Arbitrary Pattern Formation has been the subject of intensive research in the Euclidean plane. We ask readers to refer to the following excellent books [1, 2]. However, in the discrete setting, Arbitrary Pattern Formation has been considered relatively recently. The most closely related previous work is that of Bose et al. [45] solving Arbitrary Pattern Formation on a grid in the classical model. They established that each robot makes $O((\max\{D^i, D^p\})^2)$ number of moves in their algorithm, and the lower bound on the number of moves is $\Omega(\max\{D^i, D^p\})$. However, they did not provide analysis on time. Furthermore, their algorithm only works for asymmetric initial configurations. We consider both time and number of moves and both symmetric and asymmetric initial configurations. Furthermore, our algorithm is (asymptotically) optimal in both time and number of moves.

9.3. Chapter Roadmap

We discuss preliminaries in Section 9.4 and Result 1 (lower bound) in Section 9.5. Result 2 (our algorithm) is described in Section 9.6. We then conclude in Section 9.8 with a short discussion.
9.4. Preliminaries

In this section, we define the initial and target (pattern) configurations. We then define several terms that we use in the description and analysis of the algorithm.

9.4.1. Initial and Pattern Configurations

Let the initial configuration be denoted as $C_{\text{init}}$. In $C_{\text{init}}$, $n$ robots in $Q$ are positioned on $n$ distinct points of a grid. Let the pattern configuration be denoted as $C_{\text{pattern}}$. $C_{\text{pattern}}$ has $n$ positions that need to be occupied by $n$ robots forming the target pattern.

9.4.2. Leader Election

In the classical model, we do not have obstructed visibility, meaning that each robot sees all others at all times. However, Dieudonné et al. [81] has shown that Leader Election is solvable in the classical model if and only if Arbitrary Pattern Formation is solvable. That is to say, if there is no way to elect a leader, then Arbitrary Pattern Formation cannot be solved. Therefore, either Leader Election directly or some manner of symmetry breaking is required for the problem to be solvable. We use Leader Election, which uses randomization for the cases of symmetric $C_{\text{init}}$, otherwise a deterministic approach is sufficient. Particularly, we use the distance from the centroid of the current configuration as a ranking for leader. If there is a robot more than 1 unit further from the centroid than all other robots, then it becomes the leader immediately.
but if there is a tie, then the tied robots move outwards with some probability until there
is a single robot more than 1 unit further from the centroid than all other robots. A lead
of only 1 unit is not sufficient for solving randomized Leader Election in the asyn-
chronous setting. This is because a tied robot could wake up and look, decide to move but
take a long time to execute the movement. Meanwhile, the other tied robot moves and
sees that it is the farthest and declares itself leader, only for the first robot to finally move
and the two robots would be tied again. With high probability, Leader Election can
be solved using randomization in $O(\log n)$ time for $n$ robots.

9.5. Time Lower Bound

We prove the following theorem for the time lower bound of any Arbitrary
Pattern Formation algorithm (deterministic or randomized). The lower bound of
$\Omega(\max\{D^i, D^p\})$ on the number of moves by a robot is given in Bose et al. [45].

**Theorem 9.1** For an initial configuration with robots on distinct grid points, any (de-
terministic/randomized) algorithm for Arbitrary Pattern Formation in any model
(classical model with no lights or the robots with lights model) needs $\Omega(\max\{D^i, D^p\})$
rounds in the $FSYNC$ setting.

**Proof.** We prove the time lower bound in the $FSYNC$ setting without lights, which ap-
plies immediately to the $ASYNC$ setting as well as the lights model.

The lower bound construction is as follows. Consider an initial configuration $C_{init}$. Let $CIR_i$ be the smallest enclosing circle (SEC) containing all $n$ robots of $C_{init}$. Let $D^i$ and $P^i$, respectively, be the diameter and perimeter of $CIR_i$. Similarly, consider a target
configuration $C_{pattern}$. Let $CIR_p$ be the SEC containing all $n$ target points in $C_{pattern}$. Let
$D^p$ and $P^p$, respectively, be the diameter and perimeter of $CIR_p$. Furthermore, let $C^i$ ($C^p$) be the center of $CIR_i$ ($CIR_p$).

Assume that $D^p = \Omega((D^i)^{O(1)})$ and position $CIR_i$ inside $CIR_p$ such that the center $C^i$ of $CIR_i$ coincides with the center $C^p$ of $CIR_p$. Since $CIR_i$ and $CIR_p$ are SECs, there must be at least three robots on the perimeter $P^i$ of $CIR_i$ and at least three target points on the perimeter $P^p$ of $CIR_p$.

Let $L_{\text{min}}$ be the minimum distance from the perimeter of $CIR_p$ to all the robots in $CIR_i$. We have that $L_{\text{min}} \geq \frac{D^p - D^i}{2}$. Since we assumed that $D^p = \Omega((D^i)^{O(1)})$, $L_{\text{min}} \leq D^p/4$. Since at least three target points are on the perimeter $P^p$ of $CIR_p$, at least three robots from $CIR_i$ need to traverse at least distance $L_{\text{min}}$ to reach to those target points. Since a robot can move one unit on the grid in each LCM cycle, $L_{\text{min}}$ distance translates to $L_{\text{min}}$ LCM cycles. Therefore, in the $\mathcal{FSYNC}$ setting, any algorithm for $\text{Arbitrary Pattern Formation}$ needs $L_{\text{min}} \geq D^p/4$ rounds.

Now, we prove the lower bound of $D^i/4$ rounds. For this, we switch $C_{\text{init}}$ and $C_{\text{pattern}}$. This means that $CIR_p$ is inside $CIR_i$ with centers $C^i$ and $C^p$ coincide. Assume that $D^i = \Omega((D^p)^{O(1)})$. The (at least) three robots on the perimeter $P^i$ of $CIR_i$ need to move at least to the perimeter $P^p$ of $CIR_p$. The minimum distance $L'_{\text{min}} \geq \frac{D^i - D^p}{2} \geq D^i/4$ for this move. Therefore, arguing similarly as above, in the $\mathcal{FSYNC}$ setting, any algorithm for $\text{Arbitrary Pattern Formation}$ needs $L'_{\text{min}} \geq D^i/4$ rounds.

$\square$

9.6. Classical Model Algorithm

In this section, we describe an $O(\max\{D^i, D^p\})$-time, $O(\max\{D^i, D^p\})$-moves algorithm for $\text{Arbitrary Pattern Formation}$ on a grid in the $\text{ASYNC}$ setting in the...
Algorithm 2: Arbitrary Pattern Formation on a grid in the classical model

if $n$ robots match target configuration then
  | Done
else if else if $n - 1$ robots match target configuration then
  | Phase 3
else if a leader robot is selected and it is at least $10D' + 10H_p + 10W_p$ above
  | any non-leader robot not in its target column then
  | Phase 2
else
  | Phase 1

classical model (without lights). We have unobstructed visibility, i.e., the robots can determine the (relative) positions of all robots at every step; however, they do not have lights to separate states and provide a form of communication.

We use $(x_i, y_i)$ to denote the current position of robot $r_i$. We use $(x_{pi}^p, y_{pi}^p)$ to denote the target position of $r_i$ at the end of phase $p$. The target configuration of robots is $T$.

Algorithm 2 is broken into three phases, Phase 1–3. Phase 1 performs LEADER ELECTION, and the leader positions itself such that all robots share a common coordinate system, which will be maintained through Phase 2. Phase 2 moves all of the non-leader robots from their initial to their final positions. Phase 3 moves the leader into the only remaining target position, completing the pattern. An illustration of this process is provided in Fig. 9.1.

Since robots can determine the positions of all other robots, all robots agree on the current phase based on the conditions in Algorithm 2. The robots seek to form $C_{\text{pattern}}$ that has SER of width $W_p$, height $H_p$, and diameter $D_p$. Let $\text{SER}_{\text{cur}}$ denotes the the current configuration of non-leader robots and $D' = \max\{W', H'\}$ refers to the diameter of $\text{SER}_{\text{cur}}$. Note that $D'$ (in $C_{\text{init}}$, $D' = D^i$ and in $C_{\text{pattern}}$, $D' = D^p$) changes throughout the
9.6.1. Phase 1 - Elect a Leader

The goal of phase one is to select a special robot called the leader, and move it to a position such that all robots know it is the leader and it will remain so throughout subsequent phase. The robots will use the position of the leader to retain information about the current phase.

We first pick the leader using a combination of two methods. The robots attempt to elect a leader as the furthest robot from the centroid on the boundary of the SER$_{cur}$ of the current configuration of robots. If there are multiple robots who are tied for farthest from the centroid, then the tie is broken using the binary occupancy string method of Bose et al. [45]. Specifically, SER$_{cur}$ is represented using a $M \times N$-length binary string, where $M, N$ are the side lengths of SER$_{cur}$, such that 0 indicates a vacancy of a position and a 1 indicates the position is occupied by a robot. A binary string is created start-
ing from each corner of SER\textsubscript{cur} and labeling positions towards the closer of the two adjacent corners. Then we move to the next closest column and proceed in the same direction as the previous column, and this is repeated until every position in the SER\textsubscript{cur} is represented by a character in the binary string. If the SER\textsubscript{cur} is a square, then a string is created starting from each corner continuing in the direction of both adjacent corners for a total of 8 binary strings. The lexicographically largest binary string will be unique, otherwise the configuration is necessarily symmetric and thus not deterministically solvable. Randomization may be used in this case as previously described. The first ‘1’ occurring in the lexicographically largest binary occupancy string corresponds to the leader. Therefore, using the combination of these methods, any asymmetric initial configuration C\textsubscript{init} results in an agreed-upon leader.

Once a leader is chosen, orientation is determined as follows: the leader robot is the topmost robot on the leftmost column of the SER\textsubscript{cur} of the current configuration. Therefore all robots have a sense of up, down, left, and right as long as the leader remains the topmost robot of the leftmost column. The topmost robot moves up directly to height 10D\textsuperscript{'} + 10H\textsuperscript{p} + 10W\textsuperscript{p} above the highest non-leader robot. No robot had a farther distance from the centroid than the leader before it moved upwards, and it has only moved further from the centroid, therefore it will retain leadership as long as it continues to move no closer to the centroid.

**Condition 1.1:** Robot \( r_i \) is the furthest robot from the centroid on the boundary of the SER of the current configuration, \( y_i < y_1 \) and \( x_i < x_1 \)

**Action 1.1:** Robot \( r_i \) moves up towards \((x_i, y_1)\)
Condition 1.2: Robot $r_i$ is tied as the furthest robot from the centroid on the boundary of the SER of the current configuration, and is the leading robot (the first ‘1’) in the largest binary occupancy string.

Action 1.2: Robot $r_i$ moves up towards $(x_i, y_i^1)$

Note that either Condition 1.1 or Condition 1.2 must be true if $C_{\text{init}}$ is asymmetric.

**Lemma 9.2** If we have an asymmetric configuration $C_{\text{init}}$ at the start of phase 1, then all robots will agree on a leader throughout phase 1.

**Proof:** One or more robots on the boundary of the SER are, or are tied for, the furthest from the centroid. There is a uniquely largest binary occupancy string using the method previously described that will allow any ties on the largest distance from the centroid to be broken; otherwise, the configuration is not asymmetric. If there is a unique robot that is farthest from the centroid, call it the “winner.” Otherwise, the winner is the robot corresponding to the ‘1’ in the lexicographically largest binary occupancy string. The winner moves outwards from the centroid. Only the leader robot in phase 1 moves, and it moves strictly farther from the centroid throughout phase 1. Therefore the leader retains leadership throughout phase 1.

**Lemma 9.3** With a leader chosen, all robots agree on an orientation within 1 epoch.

**Proof:** If the leader is not on the corner of the SER of the current configuration, then $up$ is along the SER in the direction further from the centroid. There can be no robot in its way otherwise that robot would’ve been the leader. If the line between the centroid and the leader is perpendicular to the boundary line on the SER on which the leader lies, then it arbitrarily picks one direction to go. From there on, there is a unique direction on the boundary line moving it further from the centroid. If the leader is at the corner of the
SER, it moves in the direction to increase the size of the SER along the line coincident with the longer boundary edge. If the boundary edges are even, the leader picks one direction arbitrarily. From there on, there is a unique boundary edge that is longer.

**Lemma 9.4** Within $O(D' + H^p + W^p)$ epochs, the leader will be in position to complete phase 1.

**Proof:** Using the results of Lemma 9.3, the leader will move in one direction towards a corner of the SER$_{cur}$, with no robot blocking its path within $O(D')$ epochs, if it is not already on the corner. It will then continue in one direction expanding the SER$_{cur}$ upwards until it is $10D' + 10H^p + 10W^p$ above the topmost non-leader robot, where $D'$ is the diameter of the configuration SER$_{cur}$ of non-leader robots. No robot can block the leader as it moves further from the centroid to its goal position for Phase 1, otherwise the blocking robot would be further from the centroid, contradicting the assumption that the blocked robot was the leader. Since the leader does not have to wait for any robot as it moves out, it moves once per epoch towards $10D' + 10H^p + 10W^p = O(D' + H^p + W^p)$ above the topmost non-leader robot.

9.6.2. Phase 2 – Reposition $n - 1$ Non-leader Robots to their Final Positions

Phase 2 comprises the bulk of the algorithm. The goal of this phase is to move all of the non-leader robots from their initial positions to their final positions. Because their movements are highly coordinated, the robots perform the necessary movements to accomplish this in a single phase.

The mapping of non-leader robots to their final positions is as follows: The non-leader robots are enumerated in column-major format, i.e. top-to-bottom then left-to-
right, as are their corresponding final positions. The topmost goal position on the leftmost
column is reserved for the leader robot. At the start of Phase 2, the top left corner of the
SER\textsubscript{cur} of the configuration of non-leader robots is initially $10D' + 10H^p + 10W^p$ below
the leader. The top left corner of the SER of the goal positions of the non-leader robots is
$8H^p + 8W^p$ below the leader. The left edges of the initial and final configurations of non-
leader robots lie on the same column.

**Condition 2.1:** Robot $r_i$ is at position $(x_i, y_i)$ such that $x_i < x_i^2$, there is no robot $r_j$ at
$(x_i + 1, y_i)$, and there is no robot $r_j$ at $(x_i + 1, y_j)$ in its target column such that $y_j < y_i$

**Action 2.1:** Robot $r_i$ moves right towards $(x_i^2, y_i)$

**Condition 2.2:** Robot $r_i$ is at position $(x_i, y_i)$ such that $x_i > x_i^2$, there is no robot $r_j$ at
$(x_i - 1, y_i)$, there is no robot $r_j$ at $(x_i - 2, y_i)$ such that $x_j^2 = x_i - 1$, and there is no robot
$r_j$ at $(x_i - 1, y_j)$ in its target column such that $y_j < y_i$

**Action 2.2:** Robot $r_i$ moves left towards $(x_i^2, y_i)$

**Condition 2.3:** Robot $r_i$ is at position $(x_i^2, y_i)$ such that $y_i < y_i^2$, there is no robot $r_j$ at
$(x_i, y_i + 1)$, and there is no robot $r_j$ such that $y_j < y_i - 2H^p$

**Action 2.3:** Robot $r_i$ moves up towards $(x_i^2, y_i^2)$

**Lemma 9.5** *The leader is maintained throughout Phase 2 for $n \geq 3$ robots.*

**Proof:** The leader is initially $10D' + 10H + 10W$ above the SER of the current config-
uration of non-leader robots. Until all non-leader robots have reached their target columns,
the vertical distance between the leader and SER cannot be less than $10D' + 8H + 10W$,
since non-leader robots cannot move up past $2H$ above another non-leader robot, and
robots will not move up until reaching their target column. SER is a rectangle of shape $\max\{D', 2Hp\} \times \max\{D', Wp\}$. Thus, until all non-leader robots have reached their target columns, the non-leader robots will be no more than $\max\{D', 2Hp\} + \frac{1}{n}(10D' + 10Hp + 10Wp) + \max\{D', Wp\}$ from the centroid, while the leader will be at least $\frac{n-1}{n}(10D' + 8Hp + 10Wp)$ from the centroid, which is larger for $n \geq 3$. Therefore, until all non-leader robots have reached their target columns, the leader doesn’t change.

Once all non-leader robots have reached their target columns, the width of SER is $W$. If the height of SERn is greater than $2H$, then the leader is still $10D' + 8H + 10W$ above SERn and the leader will not change as robots move within SERn using the previous argument. If the height of SERn is no greater than $2H$, then the current leader is at least $8H + 8W$ above SERn, which is now a rectangle of maximum size $W \times 2H$. In this case, the non-leader robots will be no more than $\frac{1}{n}(8H + 8W) + W + 2H$ from the centroid, while the leaders will be at least $\frac{n-1}{n}(8H + 8W)$ from the centroid, which is larger for $n \geq 3$. Thus, the leader will not change. □

**Lemma 9.6** No deadlocks occur during phase 2.

**Proof:** If a robot is the topmost robot in its column moving up and it does not have a robot $2H$ units below it, then it will move upwards. Robots below it will then move up behind the topmost robot since they must have no robot $2H$ units below as well. Once the topmost robots reach a point where there is a robot $2H$ units below, they will wait to wait for the bottommost robots to move up. If the bottommost robots are in their target columns, then either they or some robots above them can move up since there will be no more than $H$ robots in a target column. If the bottommost robots are not in their target columns, then there is a robot moving left (resp., right) with no robot to its left (resp.,
right) between it and its target column. This robot will move towards its target column and robots behind it will then follow. A robot \( r_i \) moving horizontally cannot reach a deadlock by encountering another robot \( r_j \) on the same row to its right (resp., left) that is trying to move left (resp., right) past it, because this would contradict the assumption that robots and their target positions are numbered top to bottom then left to right. If two robots on the same row are trying to move to the same position, we arbitrarily choose the left robot to move into the column first. It will then be on its target column and begin moving upwards. The only remaining case where a robot moving horizontally will wait is if a robot moving horizontally towards its target column would place it above a robot already in its target column. Since these are the bottommost robot, as before, the robots moving up will have room to move up out of the way of the robot moving horizontally. The robot moving horizontally towards its target column will then be able to move into its target column and begin moving upwards. \( \square \)

**Lemma 9.7** Non-leader robots complete phase 2 in \( O(D' + W^p + H^p) \) epochs.

**Proof:** Within \( O(H) \) epochs, the robots already in their target columns have moved up until there is a robot \( 2H \) units below or there is a robot directly above them that can move no higher. These robots will fill at most \( H \) consecutive vertical positions.

Within \( O(D' + W) \) epochs, the bottom \( H \) robots will reach their target columns. The robots in the bottom \( H \) are free to move into their columns without disruptions, since all robots already in their target columns have moved up. All of the horizontally moving robots are able to move directly to their target columns, only waiting for robots in front of them unless a robot from left and another from the right both want to move into the same position, which is resolved within one epoch. Since there are no more than \( D' \) robots in a
row, no horizontally moving robot will wait longer than $D'$ epochs for a robot in front of it to get out of its way, then they will be free to move without waiting.

After the bottom $H$ robots have reached their target columns within $O(D' + W + H)$ epochs, all robots moving above them have either moved into their target columns as well or are waiting for robots to move up past them before being able to move into their target columns. A robot can only wait once for robots moving vertically to get out of its way before it can move horizontally because a robot will never move horizontally past a robot already in its final target column; otherwise would contradict the ordering of robots and their final target positions. Therefore, a robot may wait only $O(H)$ time for robots moving up to get out of the way if they are within the bottom $H$ robots. The size of the SER of non-leader robots decreases by $H$ every $O(H)$ epochs. Thus, in $O(D' + H + W)$ epochs, all robots will have reached their target columns.

9.6.3. Phase 3 – Repositioning Leader to its Final Position

The leader is the only remaining robot that hasn’t terminated at the start of Phase 3. It can see that $n - 1$ robots match their positions of the target configuration $C_{\text{pattern}}$, and it can use their positions to determine its goal position. The leader moves downward until reaching the proper row, then rightward until reaching its final position. The leader then terminates and the algorithm is complete.

**Condition 3.1:** Robot $r_i$ is at position $(x_i, y_i)$ such that $x_i < x_i^3$ and $y_i > y_i^3$

**Action 3.1:** Robot $r_i$ moves down towards $(x_i, y_i^3)$

**Condition 3.2:** Robot $r_i$ is at position $(x_i, y_i)$ such that $x_i < x_i^3$ and $y_i = y_i^3$

**Action 3.2:** Robot $r_i$ moves right towards $(x_i^3, y_i^3)$
Lemma 9.8  Phase 3 will be completed within $O(W^p + H^p)$ epochs.

Proof:  The leader now sees that all robots except itself match some rotation of the target configuration. The non-leader robots have terminated. As the leader robot moves into position, it may lose its sense of orientation. However, this causes no trouble, as it will move monotonically towards any position that will complete the pattern. If there is some symmetry such that two valid goal positions are equidistant, it arbitrarily picks one. There is no robot that can block the leader from reaching its final position, because it is the topmost robot of the leftmost column of the current configuration of robots and its goal position is the the topmost position of the leftmost column of the target configuration. The leftmost target column and the leftmost current column are the same. Therefore there is always at least one path to a goal position without any obstacles, e.g. downwards towards its target row then rightwards towards its target column. If another robot were along this path, it would contradict the assumption that the goal position reserved for the leader robot is the topmost position on the leftmost column of the target configuration. Because the leader can move directly to its target position without ever waiting, it moves $8H + 8W$ units to get to the SER of the target configuration, then less than $H + W$ units within the SER of the target configuration to reach its final position. □

9.6.4. Determining The Current Phase

Robots determine which phase they are in using the conditions stated in Algorithm 2. Since all robots see the positions of all other robots, every robot agrees on the current phase. Having shown that the phases work as they should individually, we now show that the entire algorithm terminates in correctly, regardless of the initial configuration. We
then provide an upper bound on time for Algorithm 2.

**Lemma 9.9** Algorithm 2 terminates correctly regardless of the initial configuration $C_{init}$.

**Proof:** We have shown correctness for the case where the algorithm progresses normally through the three phases. Suppose that the initial configuration causes the algorithm not to proceed through the typical ordering of phases from Phase 1 to Phase 3. For example, the initial configuration may meet the necessary conditions for Phase 2. The only goal of Phase 1 is to select a leader robot and move it sufficiently far from the remaining robots for the next phase to proceed properly. If the algorithm starts with a leader already robot in place, it is the same as if it had gone through Phase 1, and the robots may then proceed through the next phases and terminate normally. Next, there is the case of the initial configuration meeting the necessary conditions for Phase 3. In this case, the non-leaders already match the target configuration of non-leaders and there is a known leader robot. The non-leaders may then terminate and the leader may then proceed through Phase 3 the same as if Phase 1 and 2 had occurred. □

**Theorem 9.10** Algorithm 2 solves APF in $O(\max\{D^i, D^p\})$ epochs.

**Proof:** Robots in Phase 3 cannot return to Phase 1 or Phase 2 because this would imply that one of the non-leaders is no longer in its final position. However, the non-leaders go to their final positions then terminate. Robots in Phase 2 cannot return to Phase 1 if $\max\{W^p, H^p\} \leq D'$ because this would imply that the leader robot is at least $10D' + 10H^p + 10W^p$ above any non-leader robot not its target column, but robots need to reach their target column in order to move upwards, at which time they never leave their target column again, and the leader itself will not move in Phase 2 in this case. However, if $\max\{W^p, H^p\} > D'$, then the leader will need to move upwards as $D'$ expands which
places the robots back in Phase 1. Nevertheless, robots in this scenario will alternate between Phases 1 and 2 for $10 \max\{W^p, H^p\} = O(W^p + H^p)$ epochs, at which time $D'$ no longer grows larger and $\max\{W^p, H^p\} \leq D'$. Since $D' = O(\max\{D^i, D^p\})$ and $W^p + H^p \leq 2\max\{W^p, H^p\} \leq 2D^p$, the theorem follows.

9.7. Distance and Spatial Complexities

The maximum number of moves has previously been considered in the grid setting. In this model, the maximum number of moves performed by any robot is equivalent to the distance complexity, since each robot can only move unit distance per move. Therefore, we have the following lemma.

**Lemma 9.11** Algorithm 2 has $\Theta(\max\{D^i, D^p\})$ (and therefore, optimal) distance complexity.

As in Chapter 5, we use the area to describe the spatial complexity in the grid setting. Recall that the definition of spatial complexity divides the area (of the smallest enclosing circle) of the largest configuration during any algorithm’s execution by area occupied in the initial configuration. However, since no scaling of the target configuration is allowed, the size of the target configuration relative to the initial configuration may be arbitrarily large. Thus, the spatial complexity can be arbitrarily large. Hence, we use the area and perimeter (of the smallest enclosing rectangle) to better reflect the spatial requirements of the algorithm.

**Lemma 9.12** Algorithm 2 has $\Theta(\max\{D^i, D^p\})$ perimeter and $\Theta\left((\max\{D^i, D^p\})^2\right)$ area complexity. Therefore, both are optimal.

**Proof:** Suppose that the target configuration is much larger than the initial configura-
dition, i.e. \(D^i \ll D^p\). Further, suppose the initial configuration is a tightly packed square of \(n = 4\) robots, while the target configuration is a much larger square with the robots initially in the center. To reach any corner of the target configuration, each robot must travel \(\Omega(D^p)\). Thus, the area of the square-shaped pattern configuration is \(\Omega \left( (D^p)^2 \right)\), and the perimeter is \(\Omega(D^p)\). Recall that a robot performs \(O(\max\{D^i, D^p\})\) moves. Therefore the maximum perimeter and area are given by:

\[
Peri = O(D^i) + O(\max\{D^i, D^p\})
\]

\[
Area = \left( O(D^i) + O(\max\{D^i, D^p\}) \right)^2
\]

In the case of \(D^i \leq D^p\), \(Peri = O(D^p)\) and \(Area = O \left( (D^p)^2 \right)\). The argument is similar for the remaining case of \(D^i > D^p\). Thus, the perimeter and area complexities are optimal. □

9.8. Conclusion

In this chapter, we have presented an algorithm for placing robots in the classical model (under unobstructed visibility) on the positions specified in a target pattern to be formed starting from arbitrary initial positions of all \(n \geq 1\) robots being on distinct grid points. We have also developed a lower bound on time to solve Arbitrary Pattern Formation to establish that our algorithm is optimal in both time and number of moves complexity (the number of moves lower bound was given in [45]). We have further shown that the area and perimeter complexities are optimal. The number of moves, area, and perimeter reflect the distance and spatial complexities in this model.
Chapter 10. Concluding Remarks

In this dissertation, we have provided many algorithms with favorable performance metrics across a wide variety of problems and models. Below, we provide a summary of our contributions.

Chapter 4 introduced two new performance metrics for robot swarm algorithms, the distance and spatial complexities. These metrics were then applied to a set of convex polygon based complete visibility algorithms, and we showed that the algorithms evaluated were optimal with respect to these metrics despite having varied performance in terms of time.

Chapter 5 considered convex polygon based complete visibility on a grid by opaque robots with lights in *ASYNC*. We first provided lower bounds on time, perimeter, and area for any Convex Hull Formation algorithm on a grid (with or without lights). We then presented a framework that can be adapted on the basis of various input parameters that is capable of achieving optimal area, perimeter, and time. In total, we provided four algorithms from the framework: one that is optimal in local computation and three that are optimal in time, perimeter, and area but have varying performance in terms of local computation.

Chapter 6 introduced the problem of Doorway Egress. We provided four algorithms for point robots: One algorithm for any model variant that had optimal distance and spatial complexities but sub-optimal time, and three algorithms for point robots making use of any one of transparency, lights, or *SSYNC* that had optimal time but sub-optimal distance and spatial complexities.

Chapter 7 addresses the problem of Doorway Egress for fat robots. We provided two algorithms for fat robots making use of grid-alignment, transparency, and either lights or *SSYNC* that had optimal time but sub-optimal distance and spatial complexities. We note that the algorithm of the previous chapter (for any model variant) that had optimal distance and spatial complexities but sub-optimal time extends to fat robots as well.

Chapter 8 then provided several lemmas that lead to our conjecture that, without any beneficial assumptions, linear time is required for Doorway Egress. This would make our linear time algorithm for any model optimal in time, distance, and spatial complexities.

Chapter 9 provided a lower bound on time for Arbitrary Pattern Formation algorithms on the grid. We then provided an algorithm for transparent robots on the grid that is optimal in time, number of moves, area, and perimeter.
The contributions of this dissertation attempt to move the various robot swarm models towards more practical applications. The performance metrics used help ensure that algorithms avoid sequential processes, arbitrarily large time complexities, and inefficiencies in regards to movement and positioning. Providing algorithms that are optimal, or have more favorable performance metrics than prior algorithms, allows for greater efficiency, cost-effectiveness, and scalability in practice.

In the future, it would be interesting to provide a lower bound on time for Doorway Egress by robots with no additional capabilities (i.e., opaque robots in $\textit{ASYNC}$ without lights). As noted in Chapter 8, we strongly conjecture that this requires $\omega(1)$ time for both point and fat robots, and perhaps even $\Omega(n)$ time. This would show that we have minimized the number of additional capabilities, or assumptions, necessary for our point-robot algorithms for Doorway Egress to achieve constant time.

It would also be of interest to provide sublinear-time algorithms for Doorway Egress by opaque, fat robots with lights in $\textit{ASYNC}$ or without lights in $\textit{SSYNC}$, or to show that such algorithms require linear time.

Some directions for future work include optimizing the existing algorithms further. For example, it would be of interest to reduce the number of colors in the algorithms of robots with lights. It would also be useful to try to modify the time-optimal algorithms for Doorway Egress such that they also have optimal distance and spatial complexities.

Some more speculative directions for future work include studying three-dimensional robot swarm applications, as well as to study robots with faulty sensors, limited visibility ranges, and other limitations to the robot capabilities.
Bibliography


[45] Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. Ar-


[56] Pavan Poudel and Gokarna Sharma. Fast uniform scattering on a grid for asynchronous oblivious robots. In *International Symposium on Stabilizing, Safety, and


[67] Gokarna Sharma, Ramachandran Vaidyanathan, Jerry L. Trahan, C. Busch, and


[79] Jurek Czyzowicz, Konstantinos Georgiou, Maxime Godon, Evangelos Kranakis,


Rory Hector was born in Homewood, Alabama on December 19, 1994. In 2018, he received his B.S. in Computer Engineering from Louisiana State University with minors in Robotics Engineering and Computer Science. He began the Ph.D. program in Electrical Engineering at LSU the same year. As a graduate student, he has worked as a teaching assistant and as a research assistant before being supported by the DoD SMART Scholarship. After graduation, he will work as a civilian contractor for the Navy. From his Ph.D. studies, he has the following publications:


In addition to the publications listed, two additional papers are accepted and awaiting publication.
