Multi A(ge)nt Systems on Graphs

Michael Amir

Multi A(ge)nt Systems on Graphs

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Michael Amir

Submitted to the Senate
of the Technion — Israel Institute of Technology
Tishrei 5783 Haifa October 2022

This research was carried out under the supervision of Prof. Alfred M. Bruckstein, in the Faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Michael Amir and Alfred M Bruckstein. Probabilistic pursuits on graphs. *Theoretical Computer Science*, 2019.

Michael Amir and Alfred M. Bruckstein. Minimizing Travel in the Uniform Dispersal Problem for Robotic Sensors. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 113–121, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems, 2019. (Visited on 07/30/2022).

Michael Amir and Alfred M. Bruckstein. Fast uniform dispersion of a crash-prone swarm. In *Proceedings of Robotics: Science and Systems*, RSS '20, 2020.

Michael Amir, Noa Agmon, and Alfred M Bruckstein. A discrete model of collective marching on rings. *International Symposium Distributed Autonomous Robotic Systems*:320–334, 2021.

Dmitry Rabinovich*, Michael Amir*, and Alfred M. Bruckstein. Optimal physical sorting of mobile agents, 2021. arXiv: 2111.06284.

Michael Amir, Noa Agmon, and Alfred M. Bruckstein. A locust-inspired model of collective marching on rings. *Entropy*, 24(7):918, June 2022.

Ori Rappel*, Michael Amir*, and Alfred M. Bruckstein. Stigmergy-based, dual-layer coverage of unknown indoor regions, 2022. arXiv: 2209.08573 [cs.MA].

Michael Amir, Yigal Koifman, Yakov Bloch, Ariel Barel, and Alfred M. Bruckstein. Multi-agent distributed and decentralized geometric task allocation, 2022. arXiv: 2210.05552 [cs.MA].

An asterisk (*) denotes equal contribution.

The generous financial help of the Technion is gratefully acknowledged.

Contents

Abstract

Multi-agent systems are a fascinating, multidisciplinary field with applications to robotics, distributed systems, biology, and social dynamics. A multi-agent system is a distributed system composed of several interacting, autonomous agents that cooperate to achieve some desired outcome. The topic of this thesis is the way in which local interactions between extremely simple agents can result in desirable global states. We study this topic from two perspectives: the perspective of an observer of the natural world, and the perspective of a designer of swarm-robotic systems. In the natural world, living swarms of organisms seem to effortlessly and autonomously coordinate their motion. How do swarms of locusts converge to a single direction of motion? Why are trails of ants so straight and nice? As observers, our goal is to study the principles underlying these kinds of phenomena, learning what we can from Mother Nature's algorithms. As designers, on the other hand, our goal is to create and guarantee the performance of swarm-robotic systems. Over the course of this thesis we seek to establish that even severely myopic and computationally limited robots can be remarkably effective when working together. We shall show that, with the right local algorithm, such robots can explore unknown environments, recover from crashes, equally split workloads, and optimize traffic systems.

Almost all mathematical models in this thesis assume the agents move in a space that is finite and discrete; namely, a graph environment, where spatial locations are indicated by vertices and the ability of an agent to move between them by edges. From a theoretical standpoint, the study of these types of multi-agent models is often ad hoc, and relatively few general techniques are known. A main goal of this thesis is to highlight a number of techniques that have proven repeatedly useful in the analysis of such models, including exchangeability, coupling, potential ("Lyapunov") functions, interacting particle systems, and stationary distributions.

Chapter 1

Introduction

It is common in the field of multi-robot systems to assume each robot possesses a powerful CPU, large memory, intricate broadcast capabilities, and rich geographical information. In the natural world, however, swarms of living organisms with far more limited capabilities seem to effortlessly coordinate their actions to a degree even sophisticated computer algorithms often struggle to reproduce. Consider the way birds, or locusts, dance in the sky before suddenly converging in one direction. Consider how the lights of fireflies turn on and off at the same time without a conductor. Consider the way ants find efficient paths to sources of food, though no ant has a map of its geographical environment. These phenomena are all examples of agents enacting a simple-but-effective local algorithm that helps them attain a shared goal. As engineers, we are tempted to push the boundaries of multi-robot systems by taking advantage of more and better: computation, communication, information. Mother Nature's designs, however, must remain adaptive enough to survive millions of years of evolution across different environmental conditions and species. She cannot make state of the art demands of the agents (living creatures) that use her algorithms, and must aim for simplicity, sometimes even universality in her design. Should we not strive for the same?

Driven by such questions, the goal of this dissertation is to deepen our understanding of the way extremely simple, local interactions between mobile-robotic or living agents can result in a desirable global state. We approach this goal from two different perspectives: that of a designer or engineer wishing to control a large swarm of robots with severely limited computation and sensing capabilities, and that of an external observer of the natural world, studying the algorithmic behaviors of swarms of living organisms.

As algorithm designers, we aspire to find simple algorithms for simple robotic swarms that result in something useful on the macro scale. The algorithms are meant to work with robots that are suitable for mass-production, cheap, technologically limited, and readily disposable. We assume only that said robots can implement local, computationally trivial algorithms, similar to those enacted in nature by social insects.

Formally speaking, our assumptions are captured by the *ant-robotics paradigm*, which places severe restrictions over our robots' sensory systems, memory, and computational capabilities. Specifically, we assume:

(i) Robots are completely autonomous, and must make their own decisions based on what they currently sense.

(ii) Robots are anonymous and identical, with every robot executing the same local-sensing based algorithm.

(iii) Robots have no means of communicating outside their sensing range. Inside their sensing range, agents only communicate in simple, implicit ways such as by detecting each others' location or (in some settings) through very simple visual signals.

(iv) Robots are oblivious or almost-oblivious, meaning they have very few or no persistent states, and their current actions are determined primarily or entirely by what they currently sense.

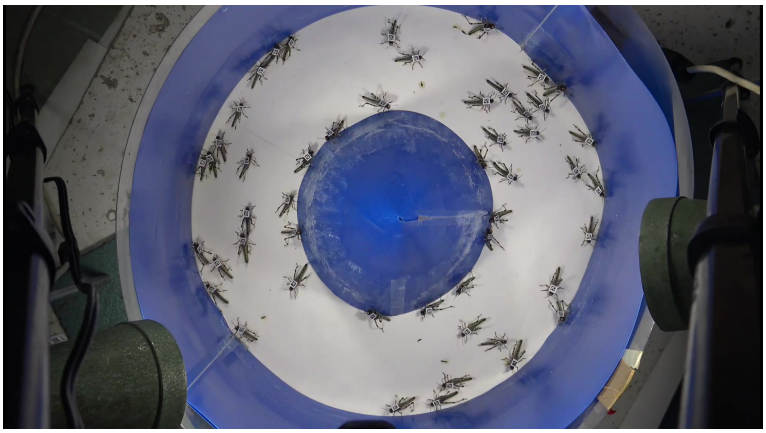
Despite these limitations, we shall show that, when instead of a single robot we have access to a huge *swarm*, even simple robots can be remarkably versatile: through coordination and the right local algorithm, they can explore unknown environments, recover from crashes, equally split workloads, optimise traffic systems, among many other applications.

The ant-robotics paradigm was originally modelled after swarms of social insects [EB13]. The paradigm is inspired by the observation that in natural swarming phenomena (such as those of ants, locusts, fireflies), centralization, long-term memory and non-spatial communication are often unnecessary. It confers two important practical advantages: first, due to the low hardware requirements, it is highly scalable, both from a systems and a mass-production perspective. Second, the ants paradigm is, by design, naturally resilient to error. The severe restrictions placed on the agents mean that the decisions robots make are relatively disentangled from each other, and so the overall robotic system can often continue to work even if some robots are displaced or crash. These advantages will be repeatedly highlighted in the many different models and algorithms we shall study.

As external observers of nature, our goal is to find and study algorithmic principles inspired by swarms of social insects–specifically ants and locusts. Here we should emphasize that the goal is not to *model* ants or locusts. The precise models underlying such insects' behaviours are very complex and subject to intense ongoing research, e.g. [AA15; AAA16; AOL$^+$14; KSA$^+$21; KAGA19]. Our goal is to look at a natural phenomenon and glean from it a simple, useful algorithm or model of behaviour, which we then analyze as a case study. For example, one thing we look at in this work (Chapter 4) is the following well-documented experiment [AAA16]: place many locusts

on a ringlike arena at random positions and orientations. They start to move around and bump into the arena's walls and into each other, and as they do so, remarkably, over time, they begin to collectively march in the same direction–either clockwise or counterclockwise (see Figure 1.1). Inspired by these experiments, we ask the following question: what are simple and reasonable myopic rules of behaviour that might lead to this phenomenon? Our goal shall be to study this question from an *algorithmic* perspective, by considering a swarm of autonomous and identical discretized mobile agents that act according to a local algorithm.

Figure 1.1 The collective clockwise marching of locusts in a ring arena (image by Amir Ayali).



These types of investigations are related to the field of *natural algorithms*, whose fundamental assertion is that the behaviour of natural organisms can be understood using concepts from the theory of robotics and computer science [Cha12; AB19a; Cha18], such as complexity analysis, look-compute-move phases, and decision-making based on discrete internal states. Natural algorithms open up interplay between biology and computer science, allowing us to study nature through the language of algorithms and vice-versa, allowing us to apply principles, algorithms and mechanisms gleaned from nature to the design of algorithms meant to service humans, such as those enacted by multi-robot systems.

Multi-A(ge)nt Systems on Graphs

We shall study swarms primarily through a theoretical lens, by proposing formal mathematical models and proving structural and/or performance guarantees within these models. Almost all mathematical models in this work assume the agents move in a space that is finite and discrete; namely, a graph environment, where spatial locations are indicated by vertices and the possibilities to move between them by edges.

The formal mathematical study of swarms, and especially swarms modelled over discrete topologies such as graphs, is often ad hoc, and very few general proof techniques are known. To this end, one goal of this dissertation is to document (and sometimes

extend) some techniques that we found repeatedly useful in the analysis of such models, including exchangeability, coupling, potential ("Lyapunov") functions, stationary distributions, and interacting particle systems. These techniques are described in Chapter Chapter 2 (Preliminaries), and their applications are highlighted wherever they occur in subsequent chapters.

Overview

Chapter 2 (Preliminaries) introduces mathematical concepts and techniques. Chapters 3-7 contain our main results. In Chapters 3 and Chapter 4 we study mathematical models inspired by two distinct natural phenomena: trails of ants and the collective marching of locusts. In Chapters 5 and 6 we discuss the *uniform dispersion problem* for robotic sensors, a fundamental problem in swarm robotics wherein we are tasked with deploying a swarm of simple autonomous robots inside an a priori unknown environment. In Chapter 7 we discuss the problem of "physical sorting," in which we are tasked with reorganizing mobile agents confined to a narrow space. Chapters 3-7 are based on the papers [AB19a; AAB21; AB20; AB19b; RAB21; AAB22]. In Chapter 8 ("Conclusion") we place our results in the broader context of swarm robotics and briefly discuss two additional, forthcoming papers that did not make it into this dissertation [RAB22; AKB+22].

Chapter 2

Preliminaries

The goal of this chapter is to give an overview of several concepts and techniques that we use in formal mathematical analysis throughout this work. As mentioned in the introduction, the formal mathematical study of multi-agent systems is often ad hoc, and very few general proof techniques are known. To this end, we found it important to document several ideas we found repeatedly useful in the analysis of such systems. These ideas have been lifted from fields such as discrete probability theory, statistical mechanics, and dynamical systems. Gathered in one place, they form a *toolbox* for studying multi-agent systems on graphs.

We keep things deliberately informal. All citations in this chapter are references to textbooks or review articles that provide additional background. Additionally, this chapter does not need to be read from cover to cover: for the readers' convenience, at the end of the chapter a table is provided (Table 2.1) documenting which techniques are used in which chapters.

2.1 A Multi-Agent Systems on Graphs Toolbox

Graphs. We begin (naturally) with the formal definition of a graph:

Definition 2.1.1. A graph is a structure made of vertices and edges that connect them. Graphs are denoted $G = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges connecting pairs of vertices (v_i, v_j). A graph is called undirected if $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$; otherwise it is called directed.

Graphs are the fundamental setting of this work. They define the environments our mobile agents inhabit. Specifically, throughout this work, the vertices of a graph represent locations, and an edge (v_i, v_j) indicates that an agent can move from v_i to v_j. For a deeper dive into graphs and graph theory we refer the reader to Gibbons, 1985 [Gib85], or to Golumbic, 2004 [Gol04a].

Markov chains. Another fundamental tool that we shall use when studying *stochastic* multi-agent systems (i.e., systems where agents make probabilistic choices) is the *Markov chain*. A Markov chain is a memoryless stochastic process: its next state depends only on its current state.

Definition 2.1.2. A Markov chain is collection of random variables $(X_t)_t$, $t = 0, 1, \ldots$, having the property that for all t, X_{t+1} depends only on X_t:

$$P(X_{t+1} = j | X_t, X_{t-1}, \ldots X_0) = P(X_{t+1} = j | X_t) \tag{2.1}$$

We assume everywhere in this dissertation that our Markov chains are *time homogenous*, which means that $P(X_{n+1} = j | X_n = i) = P(X_1 = j | X_0 = i)$.

One example of a Markov chain is the *random walk*. A random walk on a graph $\mathbb{G} = (V, E)$ is a process that begins at some vertex $v \in V$, and at each time step moves to another vertex chosen at random among the neighbors of the present vertex.

The Markov chains we discuss in this work are assumed to have a finite number of states, unless explicitly stated otherwise. Finite Markov chains are also associated with a stochastic matrix representing the probabilities of state transitions:

Definition 2.1.3. The stochastic matrix describing a Markov chain (X_t) is the matrix M whose (i, j)-th entry equals the probability of moving to state j given current state i: $M_{ij} = P(X_1 = j | X_0 = i)$.

We shall sometimes use the term "stochastic matrix" also to refer to the transpose stochastic matrix of some Markov chain, M^T. For further reading on Markov chains we defer to Asher et al., 2009 [LPW09].

We now have the language to introduce a well-known and highly useful technique for studying multi-agent systems.

Stationary distributions. Let v be some arbitrary "initialization" vector whose ith entry represents the probability of X_0 being at state i. Intuitively, vM represents the probability distribution of the process after one step, i.e., the probability distrbution of X_1. When $vM = M$, v is called a *stationary* distribution.

Definition 2.1.4. A vector π for which $\pi M = \pi$ is called a stationary distribution of the stochastic matrix M. [LPW09]

Many theorems, which we will introduce in this work as needed, state simple conditions under which Markov chains must eventually converge to a unique stationary distribution. Informally, the existence of a unique stationary distribution is useful to us because it tells us how the multi-agent system will behave at time $t = \infty$. For example, consider a graph $G = (V, E)$ where the vertex v^* contains some kind of payload. n agents are located on this graph, each of them searching for the payload by doing a

random walk until they stumble onto the vertex v^*, at which point they cease their operations and remain at v^* forever.

This simple scenario meets the definition of a Markov chain (the combined vertex locations of the agents are its states). It is difficult, and tedious, to compute the expected amount of time the agents will take to find v^*, but it is simple to prove that the unique stationary distribution of this Markov chain is the one where all agents are located at v^*, and that it converges to this distribution over time. In other words, we may not know how the multi-agent system evolves, but we know how it behaves at $t = \infty$! Throughout this work, having some notion of the state of the system at $t = \infty$ shall serve as an important "building block" for studying its behavior over time. For example, in Chapter 4 we use stationary distributions to show that a configuration of agents walking around in a ring-like arena "must" converge to the same direction of motion: knowing that it must do so in turn enables us to study the expected time it takes for this convergence to occur. In the same chapter, we also use the stationary distribution of the so-called "Discrete Heat Equation" [Law10] to prove a crucial lemma. In Chapter 3, stationary distributions are a primary object of study, and we use them to characterize the limiting behaviour of sequences of ant-like agents pursuing each other.

Coupling. In probability theory, coupling is a proof techniques that enables comparing two stochastic processes. Suppose we have two coins, the first with probability $1/2$ of turning up heads and the second with probability $2/3$ of turning up heads. Let $P_n^1(k)$ be the probability that the first coin turns up heads at least k times in n tosses, and let $P_n^2(k)$ be the probability that the second coin turns up heads at least k times in n tosses. Suppose we want to prove the intuitive claim that $P_n^1(k) \leq P_n^2(k)$. One way to do this is by direct computation - but this is a little tedious. Another way is by the following argument: let $X_1, X_2, \ldots X_n$ be indicator variables for the first coin, such that $X_i = 1$ if the ith toss of the first coin is heads, and $X_i = 0$ otherwise. Let $Y_1, Y_2, \ldots Y_n$ be a sequence of indicator variables such that if $X_i = 1$ then $Y_i = 1$ and if $X_i = 0$ then $Y_i = 1$ with probability $1/3$. We can verify that the probability that $Y_i = 1$ is precisely $2/3$, i.e., the sequence $Y_1, Y_2, \ldots Y_n$ has exactly the same distribution as a sequence of n tosses of the second coin. Furthermore, by construction $\sum_{i=1}^n X_i \leq \sum_{i=1}^n Y_i$. Since $P_n^2(k) = P(\sum_{i=1}^n Y_i \geq k)$, this implies $P_n^1(k) \leq P_n^2(k)$, as desired.

Informally speaking, the idea of *coupling* is to compare two independent stochastic processes A and B by looking at two dependent stochastic processes C and D such that C has the same distribution as A and D has the same distribution as B. In Chapter 4, coupling enables us study the evolution of a locust-like swarm of mobile agents through well-known results about random walks. In Chapter 6, a sequence of several coupling arguments enables us to compare the behavior of a robotic swarm on a graph environment $\mathbb{G}$ with that of the same robotic swarm on another graph environment, $\mathbb{G}^*$. We use this comparison to prove upper bounds on the time it takes

the swarm to complete a graph coverage mission.

For an excellent reference on probabilistic coupling we refer the reader to [Lin02].

Exchangeability. A classic riddle goes: suppose you drop n identical ants on a stick which is a meter long. Each ant begins travelling either to the left or to the right with speed 1 meter per minute. When two ants meet, they bump into each other and flip their direction of motion. When an ant reaches the end of the stick, it falls off. Can you bound the time it takes all ants to fall off in any configuration? [Win07]

Solution: assume that instead of changing direction, ants bumping into each other simply pass through each other and continue moving in the same direction. Since ants all move at the same speed, the dynamics of the system remain the same after relabelling the ants (an outside observer wouldn't be able to tell the ants are passing each other rather than flipping directions). So the longest time an ant could stay on the stick in any configuration is 1 minute.

Let us call a multi-agent system *exchangeable* if any two agents labelled A_1 and A_2 can have their locations and labels exchanged without affecting the system's evolution. The ants walking on the stick are exchangeable. Most of the swarm robotic systems we investigate in this work agents are stateless, and act only on what they currently sense, hence are exchangeable. Exchangeability is useful when two agents A_1 and A_2 are difficult to keep track of individually, but by relabelling them we can create "virtual entities" that are easier to study. In Chapter 7, we use exchangeability to create an ordered set of virtual mobile agents, and study the virtual agents instead of the original, underlying agents.

A related idea comes up in statistics: a sequence of random variables $X_1, X_2, \ldots X_n$ is called *exchangeable* if the joint probability distribution $(X_1, X_2, \ldots X_n)$ does not change when the positions of any pair of random variables X_i and X_j are exchanged (for example, $(X_1, X_2, \ldots X_n)$ and $(X_2, X_1, \ldots X_n)$ have the same distribution). Here's an example from [BMW97]: suppose an urn contains n black and m white marbles, which are drawn from an urn without replacement until the urn is empty. Let $X_i = 1$ if the ith marble is black and $X_i = 0$ if it is white. Then $X_1, \ldots X_{n+m}$ is an exchangeable sequence. This fact is used in [BMW97] to prove that the paths of agents sequentially pursuing each other on the grid are uniformly distributed. In Chapter 3 we use a novel generalization (to the best of our knowledge) of the idea of random variable exchangeability to extend the result of [BMW97] to non-grid graphs.

For further reading on statistical exchangeability we refer the readers to Aldous et al. [Ald85].

Potential ("Lyapunov") functions. Potential functions are a powerful tool for studying ant-like multi-agent systems, and dynamical systems in general. Potential functions are used to prove that a multi-agent system converges to some state $C \in \mathcal{S}$, where $\mathcal{S}$ is a set of target states. In the context of discrete multi-agent systems, the

idea of potential functions is this: let C_t denote the current state of a multi-agent system at time t. Let $F(\cdot)$ be some non-negative function over the state space such that $F(C_t) = 0$ if and only if $C_t \in \mathcal{S}$. If we can prove that $F(C_t) - F(C_{t+1}) > \varepsilon$ for some constant $\varepsilon > 0$, then after $F(C_0)/\varepsilon$ time steps the system will be in a state $C \in \mathcal{S}$.

The choice of potential function depends primarily on the set $\mathcal{S}$, which is defined based on the problem setting. When studying multi-agent consensus, a common potential function is $F(C_t) = \sum dist(A_i, A_j)$, the sum of pairwise distances between agents [BMB17]. When $F(C_t) = 0$, all agents are necessarily located in the same place, i.e., have arrived at consensus. In this example $\mathcal{S}$ is the set of all multi-agent configurations where all agents share the same location. In Chapter 4, we use potential functions to prove that two swarms of locust-like agents moving toward each other must eventually arrive at some sort of deadlock.

A variant of the potential function idea is defining a function $F(C_t)$ such that if $F(C_t) = F(C_{t+1})$ then $C_t \in \mathcal{S}$. In this variant, the fact that the multi-agent system did not progress toward something is evidence of its convergence. This variant is used in Chapter 6. In Chapter 6 we describe an algorithm that disperses agents inside a graph $\mathcal{G}$. The goal is to have at least one agent at each vertex $v \in \mathcal{G}$. The proof of the central Lemma of Chapter 6 uses potential functions called "depths" to establish that agents take longest to disperse inside a path graph of length n, hence our algorithm's time to completion on path graphs can be used to bound its time to completion on other kinds of graphs.

TASEP. The *totally asymmetric simple exclusion process* (TASEP, for short) is a paradigmatic statistical process used, among other things, to model traffic flow [CSS00] and biological transport [CMZ11]. In this process there are infinitely many agents on the integer line $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$. Each agent is activated independently infinitely often at an exponential rate of mean 1. Whenever an agent A located at x is activated, it looks at $x + 1$. If $x + 1$ contains another agent, A stays put. Otherwise A jumps to $x + 1$.

An important type of TASEP is the "TASEP with step initial condition", wherein all agents are initially placed at the non-positive integers $\{-\infty, \ldots, -2, -1, 0\}$. An important parameter in this process is the rate at which agents cross from 0 to 1. Let us write B_t to denote the number of agents that have crossed $(0, 1)$ at time t. It is shown in [Ros81] that B_t converges to $\frac{1}{4}t$ asymptotically almost surely (i.e., with probability 1 as $t \to \infty$). [Joh00] shows that the deviations are of order $t^{1/3}$. Specifically we have in the limit:

$$\lim_{t\to\infty} \mathbb{P}(B_t - \frac{t}{4} \leq 2^{-4/3} s t^{1/3}) = 1 - F_2(-s) \tag{2.2}$$

valid for all $s \in \mathbb{R}$, where F_2 is what is called the "Tracy-Widom distribution" and obeys the asymptotics $F_2(-s) = O(e^{-c_1 s^3})$ and $1 - F_2(s) = O(e^{-c_2 s^{3/2}})$ as $s \to \infty$.

Concept	Appears in
Graphs	All chapters
Markov chains and stationary distributions	Chapters 3, 4
Coupling	Chapters 4, 6
Exchangeability	Chapters 3, 7
Potential ("Lyapunov") functions	Chapters 4, 6
TASEP	Chapters 6, 7

Table 2.1: Applications of concepts and techniques from this chapter in other sections of this work.

In Chapter 6 we show that the expected time it takes a certain multi-agent algorithm to disperse n agents inside a graph $\mathcal{G}$ of size n is bounded above by the time it takes n agents to cross the $(0, 1)$ bond in TASEP with step initial condition. We then use Equation (2.2) to derive asymptotics about the algorithm's time to completion.

The TASEP model is perhaps the simplest example of a multi-agent system on a graph (the graph being $\mathbb{Z}$ with every pair of consecutive integers connected by an edge), and many of the multi-agent systems we study in this work can in some way be related to it. A variant of TASEP has agents all waking up synchronously rather than independently. A special case of a formula that we derive in Chapter 7 can be related to this variant: it gives an exact expression for the amount of time a given configuration of n synchronous agents starting at the negative integers takes to cross the $(0, 1)$ bond. Another variant of TASEP places agents on a ring of size n (that is, $\mathbb{Z}/n\mathbb{Z}$) rather than on the integers. The multi-agent model we study in Chapter 4 resembles this variant.

For an excellent introduction to TASEP, we refer the reader to [KK10].

2.2 Summary

In this chapter we drew the outline of a toolbox for studying multi-agent systems on graphs. For easy referencing, Table 2.1 outlines parts of this work where each concept is used. As evidence for the general usefulness of this toolbox, we observe that although each chapter of this work is written with a different problem and with different modelling assumptions in mind, techniques tend to be re-used between chapters. We hope that this toolbox provides a useful starting point to readers interested in studying multi-agent systems on graphs.

Chapter 3

Natural Algorithms I: Ant-like Probabilistic Pursuits on Graphs

This chapter is the first of two chapters on the topic of natural algorithms, in which we study models of multi-agent systems inspired by natural phenomena in an attempt to better understand how simple, local behaviors can lead to desirable global outcomes. The natural phenomenon inspiring this chapter is the observation that trails of ants headed toward a source of food tend to converge to efficient, straight paths, despite ants having little to no geographical information about their environment. This observation leads us to consider dynamical systems of "ant-like" agents engaged in chain pursuits and forming "ant trails" from a source location s to a target location t. We shall investigate the conditions under which the ant trails resulting from these pursuits form shortest paths from s to t.

The chapter is based on our paper [AB19a] and serves as an introduction to formal models of multi-agent systems, to Markov chains and stationary distributions, and to many interesting families of graphs. A generalization of the "exchangeability" technique mentioned in the Preliminaries is used in Section 3.4. The work [AB19a] and its precursors [Bru93; BMW97] were in many ways the initial inspiration behind this dissertation.

3.1 Introduction

Despite the myopic nature of ants, ant trails tend to be efficient, forming highly optimal paths from the ants' nest to their target destination. How can ants find such paths? A possible explanation based on the notion of "chain pursuits" is given in Bruckstein, 1993 [Bru93]. Bruckstein considers a sequence of ants emerging one after the other from a source location s. The first ant takes some arbitrary, suboptimal path to its destination, t, and stops once it reaches t. The second ant, emerging a short amount of time after the first, pursues the first ant as efficiently as it can, eventually reaching t as well. A third ant emerges shortly after the second ant and pursues the second ant,

and so on. It turns out that in this model the ants' paths converge to a straight line from s to t over time. Informally, this is because unless the ith ant walks a perfectly straight line from s to t, the $i+1$th ant will manage to slightly close the distance to the ith ant, and consequently will get to t via a slightly shorter path than the ith ant's.

In [BMW97] an analogous result is shown for a discrete model of chain pursuits over $n \times n$ grids. In this model, a sequence of agents $A_0, A_1, A_2 \ldots$ emerges from a source vertex s at times $0, \Delta, 2\Delta, \ldots$ for some fixed integer $\Delta > 1$. The agent A_0 walks an arbitrary path to the destination vertex t and subsequently stops there forever. For any $i > 0$, the agent A_i chases A_{i-1} until they both arrive and stop at t. Specifically, denote the position of A_i at time T as (x_i, y_i) and let $d_x = |x_i - x_{i-1}|$, $d_y = |y_i - y_{i-1}|$. At every time step, A_i may take a single step along either the x or the y-axis of the grid but not both. Every move that it makes must bring it closer to the current location of A_{i-1} (unless they both stand in the same place, in which case A_i does not move). Most of the time, A_i will be able to get closer to A_{i-1}'s location through both the x-axis and the y-axis. To account for this, A_i chooses, according to a probabilistic rule, whether it will move on the x or the y-axis: it moves along the x axis with probability $\frac{d_x}{d_x+d_y}$ and along the y axis with probability $\frac{d_y}{d_x+d_y}$.

It is shown in [BMW97] that when agents chase each other according to this pursuit rule, the walk of the agent A_i converges to a shortest path from s to t as i tends to infinity, irrespective of the initial path of A_0. Furthermore, the unique stationary distribution of the walks of the agents is shown to be the uniform distribution over all shortest paths from s to t. Remarkably, since in a grid graph (drawn on the plane in the usual way) the vast majority of shortest paths from s to t pass through vertices which are close to the straight line from s to t, the positions of the agents $A_0, A_1, A_2, \ldots$ will lie very close to this line almost all the time, and so the "ant trails" that the agents form on the grid will almost always look approximately like a straight line, mirroring [Bru93].

The purpose of this work is to study an extension of the model proposed in [BMW97] wherein pursuit takes place on a fixed but arbitrary graph G. Our pursuit rule is outlined in Definition 3.1.2. When the underlying graph G is a grid, this rule coincides with the pursuit rule of [BMW97] described above.

Definition 3.1.1. A walk is a sequence of vertices $v_1 v_2 \ldots v_n$ such that an edge exists between each v_i, v_{i+1}. A path is a walk where for all $1 < i, j < n$, $v_i \neq v_j$. The (vertex) length of a path or walk is the number of vertices it traverses, and is written $|P|$. For example, if $P = v_1 v_2 v_3$ then $|P| = 3$.

Definition 3.1.2 (Pursuit rule)**.** For all $i > 0$, the agent A_i pursues agent A_{i-1} by selecting, uniformly at random, a shortest path in G from its current vertex to the current location of A_{i-1}, and moving to the first vertex of that path. (The shortest path from a vertex v to itself is defined as the reflexive path $v \to v$. Hence when both A_i and A_{i-1} are located on the same vertex, A_i will stay in place).

Results. We shall show that convergence to the shortest paths in the sense of [BMW97] extends to all pseudo-modular graphs (i.e. graphs in which every three pairwise intersecting disks have a nonempty intersection), and also to environments obtained by taking graph products. Both these results include grid graphs as a special case, generalizing the result of [BMW97]. We shall also show that convergence to the shortest paths is obtained by chordal graphs (i.e. graphs in which all cycles of four or more vertices have a chord), and discuss some further positive and negative results for planar graphs. In the most general case, convergence to the shortest paths is not guaranteed, and the agents may get stuck on sets of recurrent, non-optimal walks from s to t. However, we shall show that the stationary distributions of the agents' walks will always be uniform distributions over some set of walks of equal length, generalizing [BMW97]'s result about uniform agent path distributions on the grid.

Related work. The question of whether ants need to estimate geometrical properties of the underlying surface to converge to the optimal path was posed by Feynman [Fey85] and has since been investigated in both robotics and biology, inspiring research into networks, cooperative multi-agent algorithms and dynamical systems (cf. [Bru93; BMW97; GCJT13; PSZ09; SH06]). Here we are interested in whether, when ant-like agents pursue each other using a simplistic logic that requires no persistent states and only local information regarding the environment, the "trails" they traverse on the graph converge to the shortest paths from the source vertex s to the destination vertex t. We are also interested in the probability distribution of these trails as time goes to infinity.

Various notions of pursuit have been extensively investigated for graphs under the subject of "cops and robber" games, where one or more agents attempt to capture a moving target (see [BN11] for a general survey). A greedy pursuit rule was investigated in e.g. [IK08]. Our objective is completely different, as we are instead interested in the structure over time of the trails formed by configurations of agents in pursuit of each other.

3.2 Preliminary Characterizations

In the chain pursuit model over graphs, we are given an undirected, finite, reflexive (meaning the edge (v, v) exists for every vertex v) graph G and two vertices $s, t \in G$ (not necessarily different). The vertex s will be called the source vertex, and t the destination vertex. Ants (a(ge)nts) emerge at s and pursue the ants that left before them as they walk towards the destination vertex (where they shall stop). Specifically, the first ant, A_0, follows an arbitrary finite walk from s to t, taking one step across an edge of this walk every unit of time. Furthermore, every Δ units of time - for some parameter Δ - the ant A_i leaves s and pursues ant A_{i-1} (taking one step per time unit) according to the pursuit rule described in Definition 3.1.2.

Here, Δ ("delay time") is an important parameter, and we will always assume that it is greater than 1. The pursuit rule guarantees that the distance between A_i and A_{i-1} is bounded by Δ. This is because the distance is initially at most Δ, and at every time step A_i will close the distance to A_{i-1}'s location by one vertex and A_{i-1} will move away from that location by at most one vertex, so the distance between them (so long as it is above 0) is either preserved or shortened. By the same reasoning, whenever there is a point in the pursuit where $d(A_i, A_{i-1}) = x > 0$, the distance between the ants will never subsequently increase above x. The one exception is when $d(A_i, A_{i-1}) = 0$ and both ants haven't yet stopped at t. In such cases, the distance might increase to 1 after one time step (as A_i will stay in the same spot but A_{i-1} might step to another vertex), but it will remain at most 1 from there on (note that since $\Delta > 1$ this does not contradict the earlier bound on the distance). If the distance $d(A_i, A_{i-1})$ is x once A_{i-1} reaches t, then A_i will arrive at t after x subsequent steps. Since A_{i-1} arrived at s precisely Δ time steps before A_i, this means that A_i will arrive at t in $\Delta - x \geq 0$ less steps than A_{i-1}. Consequently, the walk lengths of subsequent ants in the chain pursuit are non-increasing.

Note that in order to carry out the pursuit rule, every ant needs only local information about the graph (it need only know the disk of radius Δ about its current vertex). In Chapter 1 we discussed the *ants paradigm* in swarm robotics, which assumes robots in the swarm are identical and oblivious, and can only act based on local sensing. As expected of a multi-agent system inspired by nature, if we were to implement our chain pursuit model in a robotic swarm, it would fit perfectly within the assumptions of this paradigm.

We wish to understand the lengths of the walks of the ant A_i as $i \to \infty$, and their eventual distribution, assuming an arbitrary initial walk for ant A_0. In particular, we want to know if, in finite expected time, the walk of A_i will be an optimal path from s to t.

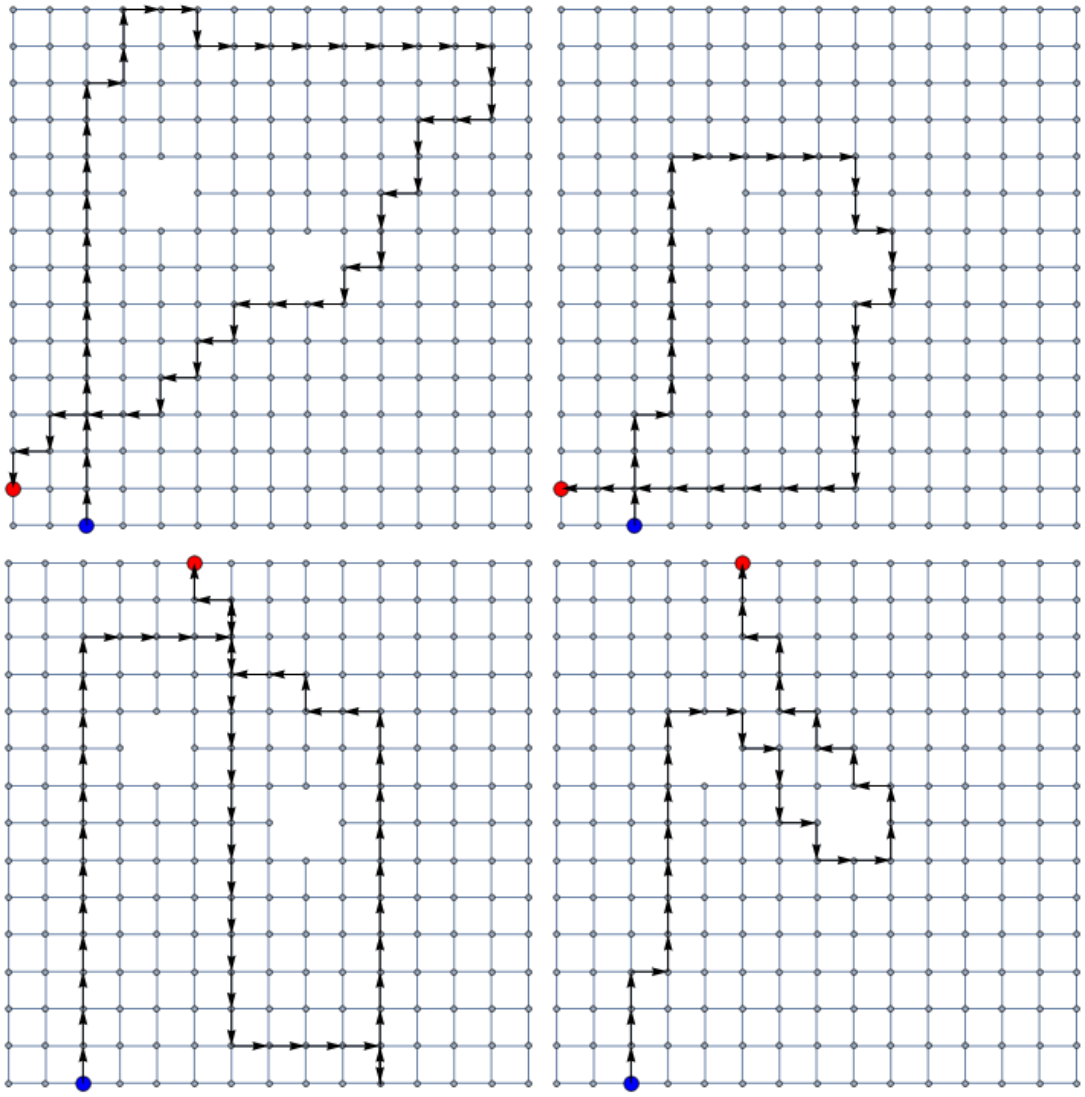
We denote the graph walk taken by A_i as $P(A_i)$. To model the distribution of ant walks over time we consider a Markov chain $\mathcal{M}_\Delta(s,t)$, parametrized by the vertices s, t and a positive integer Δ, whose states are all (the infinitely many) possible walks from s to t. The transition probability from P_1 to P_2 is defined as $Prob[P(A_{i+1}) = P_2 | P(A_i) = P_1]$, assuming the given value of Δ.

In this work we are primarily interested in studying the closed communicating classes of $\mathcal{M}_\Delta(s,t)$ (closed classes for short). A closed communicating class is a subset of states of $\mathcal{M}_\Delta(s,t)$ such that every two states communicate with each other, and no state in the set communicates with a state outside the set. A state P_i is said to communicate with state P_j if it is possible, with probability greater than 0, for the chain to transition from P_i to P_j in a finite number of steps (the reader may find more detail in the first chapter of [LPW09], or in [MT93]).

Consider the (finitely many) walks and closed communicating classes of $\mathcal{M}_\Delta(s,t)$ reachable from the initial state $P(A_0)$. Once an ant takes a walk that belongs to a closed

communicating class, the walk choices of all ants that emerge in the future will belong to this class. Additionally, since the length of $P(A_0)$ is finite and ants cannot transition to a walk of length greater than this, the walks of our ants become such that they belong to a closed communicating class in finite expected time (see [LPW09]). Hence, closed communicating classes capture the notion of "stabilization" in our dynamical system.

Figure 3.1 The behavior over time of chain pursuit over a grid graph with two "holes".



In Figure 3.1 we show the behavior over time of our dynamical system when $\Delta = 2$ over a grid with "holes". Each image illustrates the walk taken by some A_i (here we show the walk of every agent individually, though in an actual simulation, multiple ants would be walking on the graph concurrently). The image to the left shows the walk taken by A_0, and the image to the right shows a walk belonging to the closed communicating class at which the pursuit stabilized. As we can see, though the ants typically manage to shorten the walk of the initial agent $P(A_0)$ over time, they are by

no means guaranteed to converge to an optimal path.

In fact, for the particular graph environment of Figure 3.1 (a grid with "holes"), there exist an infinite number of closed communicating classes, containing walks of arbitrary length. The loops of the walk $P(A_0)$ around each of the holes determines the kinds of walks the ants may converge to.

Nevertheless, closed communicating classes $\mathcal{C}$ have some nice regularity properties. As an initial observation we will show that the walks belonging to a closed class must all have the same length, i.e. traverse the same number of vertices, denoted by $|P|$.

Lemma 3.2.1. *Let $\mathcal{C}$ be the set of walks in a closed class of $\mathcal{M}_\Delta(s,t)$. Then for all $P_l, P_k \in \mathcal{C}$, $|P_l| = |P_k|$.*

Proof Assume that there are two walks in C such that $|P_l| < |P_k|$. The walk P_l is a reachable state belonging to the closed class. However, once an ant takes walk P_l, it will never take P_k again since by the pursuit rule, walk lengths are monotonically non-increasing. Thus P_k is not recurrent in the class, a contradiction. ■

Of particular interest are closed classes that contain the shortest paths from s to t, or a subset of these paths. In [BMW97] it was shown that when G is a grid graph, $\mathcal{M}_\Delta(s,t)$ has a unique closed class containing all the shortest paths. Later we will see that whenever $\mathcal{M}$ has a unique closed class, it is necessarily the class of all shortest paths.

A concept that will be used in our arguments is *δ-optimality*:

Definition 3.2.2. We will say that a walk $P = v_1 v_2 ... v_n$ is δ-optimal, if, for any two vertices $v_i, v_{i+\delta} \in P$ we have that $d_G(v_i, v_{i+\delta}) = \delta$ (where $d_G(u,v)$ denotes the distance between two vertices in G).

We note that $d_G(v_i, v_{i+\delta}) = \delta$ implies that $v_i v_{i+1} \dots v_{i+\delta}$ is a shortest path from v_i to $v_{i+\delta}$. Therefore we have by extension that $d_G(v_i, v_{i+k}) = k$ for any $k \leq \delta$.

An important observation is that walks which belong to a closed class $\mathcal{C}$ of $\mathcal{M}_\Delta(s,t)$ must be Δ-optimal:

Lemma 3.2.3. *Let $\mathcal{C}$ be the set of walks of some closed class of $\mathcal{M}_\Delta(s,t)$. Then any walk in $\mathcal{C}$ is Δ-optimal.*

Proof Suppose for contradiction that there is some walk $P \in \mathcal{C}$ that is not Δ-optimal. Since we are in a closed class, this walk is recurrent. Thus after finite expected time some ant A_i will take walk P. After Δ time, an ant A_{i+1} will start chasing A_i via some path in $\mathcal{C}$, maintaining a distance of Δ or less from A_i at every time step.

If A_{i+1} ever manages to decrease the distance to A_i below Δ, then it will arrive at t in less steps than A_i, contradicting Lemma 3.2.1. Hence, we will assume that the initial distance between A_{i+1} and A_i is Δ. We shall show that there exists a legal "pursuit strategy" for ant A_{i+1} that can occur with non-zero probability, and will cause the

distance between A_i and A_{i+1} to drop below Δ. This leads to a contradiction (due to Lemma 3.2.1).

Since A_i follows a non-Δ-optimal walk, there exists a vertex v_l along P, such that $dist(v_l, v_{l+\Delta}) < \Delta$. Assume that l is the minimal index for which this occurs. Since l is minimal, by the pursuit rule, A_{i+1} will with some probability pursue A_i using the vertices $v_1, v_2 \dots v_l$.

Once ant A_{i+1} arrives at v_l, A_i will be at $v_{l+\Delta}$, having walked from v_l to $v_{l+\Delta}$ along the vertices of $P(A_i) = P$. Consequently, at this point in time, we will have that $d(A_i, A_{i+1}) = d(v_l, v_{l+\Delta}) < \Delta$. Since A_{i+1} has successfully dropped the distance between itself and A_i below Δ, which was the distance between them at the moment A_{i+1} emerged from s, the walk $P(A_{i+1})$ must be shorter than $P(A_i)$. This directly contradicts Lemma 3.2.1. ■

We will be primarily interested in graphs G for which closed classes contain only shortest paths from s to t; in other words, graphs on which convergence to a shortest path is guaranteed. We will consider as a special case graphs for which there is a unique closed class which contains *all* shortest paths.

Definition 3.2.4 (Convergent graphs)**.** Let $\mathcal{M}_\Delta(s,t)$ be a Markov chain defined over the graph G. If all closed classes in $\mathcal{M}_\Delta(s,t)$ contain only shortest paths from s to t, then G is called (s,t)-**convergent** with respect to delay time Δ. When this holds for *all* pairs of vertices (s,t), and any $\Delta > 1$, G is called *convergent*.

Definition 3.2.5 (Stable graphs)**.** If, for a fixed Δ, $\mathcal{M}_\Delta(s,t)$ has a unique closed class, then G is called (s,t)-**stable** with respect to delay time Δ. When this holds for *all* pairs of vertices (s,t), and any $\Delta > 1$, the graph G is called *stable*.

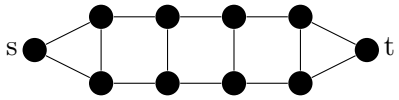
In every Markov chain $\mathcal{M}_\Delta(s,t)$ over any graph G there is at least one closed class that contains a shortest path (since walk lengths are monotonically non-increasing, and since in G there is a shortest path from s to t and we can set that path to be $P(A_0)$). Thus if G is stable, the unique closed class of $\mathcal{M}_\Delta(s,t)$ contains only shortest paths from s to t. Thus a stable graph is in particular a convergent graph:

Proposition 3.2.6. *If G is (s,t)-stable with respect to Δ, then it is (s,t)-convergent with respect to Δ.*

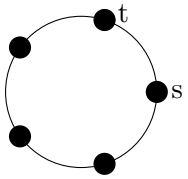
In fact, as will be shown, stable graphs are precisely the graphs for which the walks of the ants converge to a unique stationary distribution over *all* shortest paths from s to t.

An example of a graph which is convergent but not stable is given in Figure 3.2, (a). It can be proven simply and directly that it is convergent, with the tools developed later in this section. It is not stable, since if A_0 takes the top shortest path from s to t no subsequent A_i will ever use the bottom shortest path.

Figure 3.2 Examples of graphs which are not convergent or not stable.



(a) A convergent graph that is not a stable graph.



(b) C_5: A graph which is not convergent.

The simplest example of a graph that is not convergent (so also not stable) is the cycle on n vertices C_n for $n \geq 5$ (Figure 3.2, b). In fact, this graph has an infinite amount of closed classes for Markov chains with $\Delta = 2$ formed by looping clockwise from s to t and back to s any number of times, where s and t are selected to be two adjacent vertices. It is not difficult to come up with other graphs where convergence fails (these are typically, but not necessarily, sparse graphs with big cycles) - some interesting examples are provided in Section 3.3.3.

The value of Δ is meaningful when discussing (s,t)-stability or convergence. For instance, the 5-cycle in Figure 3.2, (b) is (s,t)-convergent with respect to $\Delta = 3$, but not $\Delta = 2$.

Our first characterization of stable and convergent graphs will be in terms of local "deformations" of walks to one another.

Definition 3.2.7 (δ-deformability)**.** 1. Let $P = U_1U_2U_3$ and $P^* = U_1U_2'U_3$ be two walks from v_1 to v_n, such that U_i is a (possibly empty) sub-walk in G, and such that $|U_2'| \leq |U_2| \leq \delta - 1$. Then P^* is said to be an *atomic* δ-deformation of P.

2. If there is a sequence of atomic δ-deformations of P that results in P', then P' is said to be a δ-deformation of P.

A δ-deformation of P can result in a walk of shorter length (by replacing U_2 with a shorter or even empty sub-walk), but will never lengthen it. Atomic 2-deformations are illustrated in Figure 3.3. Another example can be seen in Figure 3.8, (b), later on.

Intuitively, an atomic δ-deformation of a walk P is a *small local change* (replacing at most $\delta - 1$ vertices) of P. We wish to prove the following:

Proposition 3.2.8. *G is (s,t)-convergent with respect to Δ if and only if every walk from s to t is Δ-deformable to a shortest path from s to t.*

Figure 3.3 Illustration of a 2-deformation with and without walk shortening. Illustrated are walks drawn on the 4-cycle and 3-cycle graphs. On the left, we deform the path $v_1v_2v_3$ to $v_1v_2'v_3$. On the right, we deform $v_1v_2v_3$ to v_1v_3.

Proposition 3.2.9. *G is (s,t)-stable if and only if there exists a fixed shortest path P from s to t such that every walk from s to t is Δ-deformable to P.*

Proposition 3.2.8 can be interpreted as saying that if any walk P is transformable to a shortest path by a sequence of small, local Δ-changes to its vertices, then our ants are guaranteed to find a shortest path to their destination (in finite expected time), and vice-versa.

We require some lemmas to prove these propositions.

Lemma 3.2.10. *If P is a walk belonging to a closed communicating class of $\mathcal{M}_\Delta(s,t)$, then so is any Δ-deformation of P.*

Proof First we prove the statement for the case of atomic Δ-deformations. Let $P' = U_1U_2'U_3$ be an atomic Δ-deformation of $P = U_1U_2U_3$. Suppose A_i takes walk P. We will show A_{i+1} can take walk P' with non-zero probability.

We note that since P is Δ-optimal (due to belonging to a closed class), we must have precisely $|U_2'| = |U_2|$ (otherwise the Δ-optimality of P would break at the last vertex of U_1). Therefore we also have that $|P| = |P'|$.

Let $P = u_1u_2 \dots u_n$ and $P' = u_1 \dots u_j u'_{j+1} \dots u'_{j+\Delta-1} u_{j+\Delta} \dots u_n$, such that $U_2' = u'_{j+1} \dots u'_{j+\Delta-1}$. It suffices to show that if A_{i+1} is standing on the kth vertex of P' and A_i on the $(k+\Delta)$th vertex of P, then A_{i+1} may move to the $(k+1)$th vertex of P' in the next time step in pursuit of A_i. We show this by separation to cases:

1. If A_{i+1} is standing on u_k, for $1 \leq k < j$, then A_i is standing on $u_{k+\Delta}$, and by the Δ-optimality of P we have that $d(u_k, u_{k+\Delta}) = \Delta$, implying also that $d(u_{k+1}, u_{k+\Delta}) = \Delta - 1$. Hence, A_{i+1} may with some probability move to u_{k+1}, due to the pursuit rule.

2. If A_{i+1} is standing on u_j, then since $|U_2'| = |U_2|$ we have that $d(u'_{j+1}, u_{j+\Delta}) \leq \Delta - 1$ (in fact this is an equality). Furthermore, similar to (1) we have that $d(u_j, u_{j+\Delta}) = \Delta$, so we may have A_{i+1} move to u'_{j+1} in the next time step.

3. If A_{i+1} is standing on u'_k for $j < k < j + \Delta$, then since $|U_2| = |U_2'|$ we have that $d(u'_k, u_{k+\Delta}) \leq \Delta$. However, since P belongs to a closed communicating class,

this inequality cannot be strict (otherwise A_{i+1} would've been able to lower its overall walk length below $|P(A_i)|$). Therefore we have that $d(u'_k, u_{k+\Delta}) = \Delta$. Furthermore due to the indices and the fact that P' is a walk we have that $d(u'_{k+1}, u_{k+\Delta}) \leq \Delta - 1$, so A_{i+1} may move to u'_{k+1} (or u_{k+1}, if $k = \Delta - 1$) in pursuit of A_i.

4. If $j + \Delta \leq k$, then the proof proceeds similar to (1).

This shows that A_{i+1} may always, with some probability, pursue A_i by taking the next vertex of P', and this in turn shows that P' belongs to the same closed class as P. Since any Δ-deformation of P can be constructed from a sequence of atomic deformations, the proof is complete. ■

Lemma 3.2.11. *For a Δ-delay chain pursuit, we have that: for all i, $P(A_{i+1})$ is a Δ-deformation of $P(A_i)$.*

Proof We will define a sequence $P_0, P_1, P_2, \ldots P_N$ of atomic Δ-deformations that deform $P(A_i)$ to $P(A_{i+1})$, such that $P_0 = P(A_i)$ and $P_N = P(A_{i+1})$. P_k is defined recursively, based on P_{k-1}.

Write $P_0 = v_1 \ldots v_{n_1}$ and $P_N = u_1 \ldots u_{n_2}$, where $v_1 = u_1$ and $v_{n_1} = u_{n_2}$. Note that by the pursuit rule we have for all r that $d(u_r, v_{r+\Delta}) \leq \Delta$ and that u_{r+1} lies on a shortest path from u_r to $v_{r+\Delta}$. Thus there is always a shortest path from u_r to $v_{r+\Delta}$ passing through at most $\Delta - 1$ vertices, starting with u_{r+1}.

For a given k, P_k is a Δ-deformation of P_{k-1}, replacing the sub-walk $u_k \ldots v_{k+\Delta}$ with a shortest path from u_k to $v_{k+\Delta}$ passing through u_{k+1} (if $k + \Delta > n$ we set $v_{k+\Delta} = v_n$).

An example of the sequence for $\Delta = 3$ and $|P_0| = |P_N| = 6$ is seen below. x_i is an arbitrary vertex along a shortest path, as constrained by the definition of P_k.

$$\begin{aligned}
&\mathrm{v}_1\mathrm{v}_2\mathrm{v}_3\mathrm{v}_4\mathrm{v}_5\mathrm{v}_6 \overset{3-def.}{\rightsquigarrow}\\
&\mathrm{v}_1\mathbf{u}_2\mathbf{x}_3\mathrm{v}_4\mathrm{v}_5\mathrm{v}_6 \rightsquigarrow\\
&\mathrm{v}_1\mathrm{u}_2\mathbf{u}_3\mathbf{x}_4\mathrm{v}_5\mathrm{v}_6 \rightsquigarrow\\
&\mathrm{v}_1\mathrm{u}_2\mathrm{u}_3\mathbf{u}_4\mathbf{x}_5\mathrm{v}_6 \rightsquigarrow\\
&\mathrm{u}_1\mathrm{u}_2\mathrm{u}_3\mathrm{u}_4\mathbf{u}_5\mathrm{u}_6
\end{aligned}$$

The following is always true:

1. P_k is a valid walk in G.

2. P_k is always a Δ-deformation of P_{k-1}, as it replaces at most $\Delta - 1$ vertices (note that P_1 is well-defined since $u_1 = v_1$)

3. The first k vertices of P_k match those of P_N.

So we see that for some $N < n_2$, we will have $P_N = P(A_{i+1})$, completing the proof. ■

An important corollary of the Lemmas just proven is that a closed communicating class $\mathcal{C}$ is equal, precisely, to the closure under Δ-deformations of any walk $P \in \mathcal{C}$.

To prove Proposition 3.2.8 we apply the lemmas. In one direction, assume that the graph is convergent. Then for every walk P there is a valid sequence of walks taken by successive ants (the first ant taking walk P) that goes to a shortest path. By Lemma 3.2.11 this means that every walk can be Δ-deformed to a shortest path. The other direction follows from lemma 3.2.10, and due to the fact that the walks of the ants will enter some closed class of $\mathcal{M}$ in finite expected time. ■

To prove Proposition 3.2.9, in one direction, suppose that the graph is stable. So it has a unique closed class. Recall that every stable graph is a convergent graph. Due to Proposition 3.2.8 and 3.2.11, this implies that every walk is Δ-deformable to a shortest path from the unique closed class. All paths in this class are deformable to each other due to the mentioned closure, so it follows that every walk is deformable to a fixed (shortest) path. In the other direction, suppose every walk is deformable to the same shortest path P. Then it immediately follows from 3.2.10 that there is just one closed class, so the graph is stable. ■

Earlier we mentioned that the unique closed class of $\mathcal{M}_\Delta(s,t)$ when G is stable necessarily contains *all* shortest paths from s to t. We can now prove this. To start, we know the unique closed class, that we will denote $\mathcal{C}$, can contain only shortest paths. Now let P be a shortest path not in $\mathcal{C}$. Since the successors of any agent following this path must eventually (in expected finite time) end up taking a path in $\mathcal{C}$, it follows from Lemma 3.2.11 that there is a sequence of atomic Δ-deformations of P - each deforming it necessarily to another shortest path - such that at the end of this sequence is a shortest path belonging to $\mathcal{C}$. Let P' be the penultimate path in this sequence, that is, the last one not belonging to $\mathcal{C}$. Any Δ-deformation P^* of P' cannot shorten it (as it is a shortest path), thus it necessarily replaces a sub-walk of length $k \leq \Delta - 1$ in P' with a different sub-walk of length k. But by doing the reverse we can Δ-deform P^* back to P', thus P' belongs to $\mathcal{C}$ - contradiction. This gives a stronger characterization of stable graphs:

Proposition 3.2.12. *G is (s,t)-stable (with respect to Δ), iff its unique closed class contains all shortest paths from s to t.*

$\mathcal{M}_\Delta(s,t)$ has a unique closed class, and it is easy to see that it is aperiodic when restricted to this class (since any shortest path taken by A_i has a chance of repeating itself for A_{i+1}). Hence it has a unique stationary distribution [LPW09]. Proposition 3.2.12 implies that $\mathcal{M}_\Delta(s,t)$ has a unique stationary distribution if and only if it has a unique stationary distribution over all shortest paths from s to t. In Section 3.4 we will show that this distribution must in fact be the *uniform* distribution over all shortest paths from s to t.

In some sense the delay time $\Delta = 2$ is a benchmark for whether a graph is convergent (resp. stable):

Proposition 3.2.13. *G is convergent if and only if it is (s,t)-convergent for any pair of vertices (s,t), with respect to $\Delta = 2$.*

Proposition 3.2.14. *G is stable if and only if it is (s,t)-stable for any pair of vertices (s,t), with respect to $\Delta = 2$.*

Both these propositions follow immediately from the fact that any 2-deformation is in particular a Δ-deformation for all $\Delta > 2$. Thus any graph which is convergent (resp. stable) with respect to $\Delta = 2$ is also convergent (resp. stable) for $\Delta > 2$. ■

In other words, we need only prove that a graph is convergent (stable) for $\Delta = 2$ to show that it is convergent (stable) for any $\Delta > 1$. The significance of this is that we can now consider stability and convergence as properties of graphs rather than properties of the dynamical system defined by our pursuit rule. As a consequence of the statements we have proved in this section, we can forget the pursuit rule and consider only the relations between walks that exist in G. Hence in the following sections, **we will assume that** $\Delta = 2$, unless stated otherwise.

3.3 Convergent and Stable Graphs

Having set up some helpful propositions in the previous section, we can begin to discuss several interesting classifications of stable and convergent graphs. In [BMW97] it was shown that the grid is stable. We focus on generalizing this result to broad classes of graphs that include the grid as a special case.

We want to study graphs whose δ-deformation can easily be understood. We find it fruitful to study graphs whose induced distance metric has certain kinds of constraints placed on it (intuitively, constraints that relate to the idea of a simply connected or convex space or to operations that preserve these). To this end we show that (i) all pseudo-modular graphs are stable, and that (ii) the stable- and convergent-graph properties are preserved under taking graph products. We then move to a discussion of chordal and planar graphs.

We state the following definition and lemma, which will become useful in several sections:

Definition 3.3.1. Let $P = v_1 \dots v_n$ be a walk in G. We call two vertices v_i, v_j *discrepancy vertices* of P, if they minimize the difference of indexes $j - i$ under the constraint that $j - i > d(v_i, v_j)$.

If P is the shortest path from v_1 to v_n, then P has no discrepancy vertices. On the other hand any non-optimal walk must have at least one pair of such vertices. We note that due to Δ-optimality, if P belongs to a closed communicating class of $\mathcal{M}_\Delta(s,t)$,

then for any pair of discrepancy vertices v_i, v_j, $j-i$ must be larger than Δ; in particular, under our assumption that $\Delta = 2$ we always have $j - i > 2$.

Lemma 3.3.2. *Let v_i, v_j be discrepancy vertices in some walk $P = v_1 v_2 \dots v_n$. Then:*

1. *There is no vertex v in the sub-walk $v_{i+1} v_{i+2} \dots v_{j-1}$ such that $d(v_i, v) + d(v, v_j) = d(v_i, v_j)$. (That is, no vertex between v_i and v_j in P belongs on a shortest path between them, or is equal to one of them).*

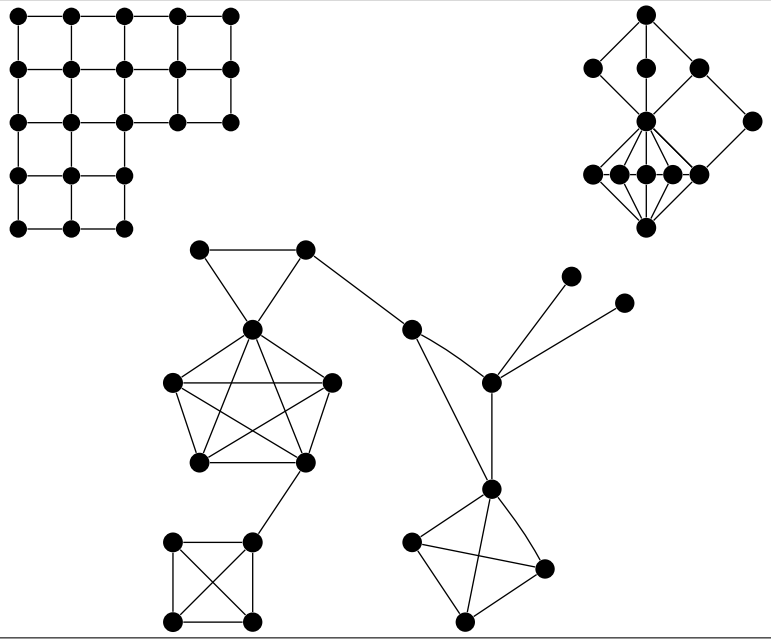
2. $j - i \leq d(v_i, v_j) + 2$

Proof For proof of (1), we note that had there been such a vertex, say $v_t \in P$, then either v_i, v_t or v_t, v_j would've been discrepancy vertices instead of v_i, v_j since the difference of indexes is smaller.

For (2), assume that $j - i > d(v_i, v_j) + 2$. From this we have: $j - (i+1) > d(v_i, v_j) + 1 \geq d(v_{i+1}, v_j)$, thus v_{i+1}, v_j are discrepancy vertices - a contradiction to the minimality property of discrepancy vertices, since $j - (i+1) < j - i$. ■

3.3.1 Pseudo-modular Graphs

Figure 3.4 Pseudo-modular graphs.



It is known that chain pursuit in the continuous Euclidean plane converges to the shortest path (see [Bru93]). One of the primary reasons for this is that the plane has no holes–it is simply connected. In a simply connected subspace of the plane, any path can be "optimized" into a shortest path by making small local deformations to it, as these deformations are never prevented by the undue presence of holes. When working with graphs, we can capture the notion of making small local changes to pursuit walks

via the Δ-deformations (and this led to propositions 3.2.8 and 3.2.9), but capturing the notion of "holes" in the right way is trickier, leading us to consider properties that are more indirect. One idea is to consider properties of convex shapes, as any convex shape is in particular holeless.

Helly's Theorem is a theorem about intersections of convex shapes, stated as follows: let $X_1 \dots X_n$ be a collection of n convex, finite subsets of R^d. If the intersection of every $d+1$ of these sets is nonempty, then the collection has a nonempty intersection (see [DGK63] and Chapter 1 of [Mat02] for additional background). Helly's theorem motivates one of the possible, equivalent definitions of pseudo-modular graphs:

Definition 3.3.3. A graph G is called *pseudo-modular*, or "3-Helly", if any three pairwise intersecting disks of G have a nonempty intersection. (A disk of radius r about the vertex v is the set of all vertices of distance $\leq r$ from v)

Pseudo-modular graphs were introduced in [BM86] as a generalization of several important classes of graphs in metric graph theory, such as the so-called "median", "modular" and "distance-hereditary" graphs (see [BC08] for a general survey). It is not hard to confirm that the grid graph is pseudo-modular, and so are many of its sub-graphs and many grid-like graphs.

To begin we wish to prove the following:

Proposition 3.3.4. *Pseudo-modular graphs are convergent.*

Proof By 3.2.13 it suffices to show that G is convergent for delay time $\Delta = 2$ (indeed, for all of our proofs from this point onwards, we will implicitly assume that $\Delta = 2$ unless stated otherwise). We show that any walk can be 2-deformed to a shortest path via 2-deformations.

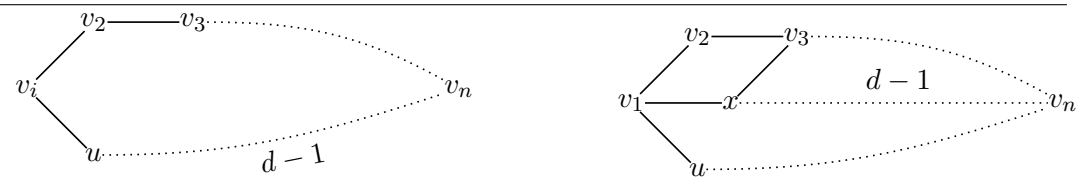
The proof is by induction on the number of vertices in the walk, denoted by n. For the induction base, it is simple to see that any walk of length 3 or less (i.e. with 3 vertices or less) from s to t can be 2-deformed to a shortest path (it is either the shortest path, or a direct link exists from s to t, and then 2-deformation clearly leads to it!). Thus the statement holds for $n \leq 3$.

Now assume that all walks of length $n-1$ can be 2-deformed to an optimal path. Let $P = v_1 \dots v_n$ be a walk with n vertices. We will show P can be 2-deformed to a shortest path.

First, as the sub-walk $v_2 \dots v_n$ is of length $n-1$, we may 2-deform it to a shortest path. Assume wlog that it already is. Then either P is a shortest path (and we are done), or v_1 and v_q, for some $q > 1$, are discrepancy vertices. If $q \neq n$ then we can 2-deform the sub-walk from v_1 to v_q into a shortest path (due to the inductive assumption), which shortens P, and reduces us to a previous case of the induction. So we simply need to handle the case where v_1 and v_n are discrepancy vertices.

If $v_1 = v_n$ (i.e. P is a "loop" around v_1) then it follows that $|P| = 3$, so we are reduced to an earlier case of the induction. Otherwise, write $d = d(v_1, v_n)$. Consider

Figure 3.5 The constructions in the proof of Proposition 3.3.4



(a) **(b)**

the three disks $\mathcal{D}(v_1, 1)$, $\mathcal{D}(v_n, d-1)$, $\mathcal{D}(v_3, 1)$, where $\mathcal{D}(v, r)$ is the disk of radius r about vertex v.

We show that the three disks intersect pairwise:

1. $\mathcal{D}(v_1, 1)$ and $\mathcal{D}(v_3, 1)$ intersect at v_2.

2. $\mathcal{D}(v_1, 1)$ and $\mathcal{D}(v_n, d-1)$ intersect at a vertex u that lies along the shortest path from v_1 to v_n.

3. By Lemma 3.3.2, 2 we have that $n - 1 \leq d + 2$, and therefore $d(v_4, v_n) \leq n - 4 \leq d - 1$. Hence $\mathcal{D}(v_3, 1)$ and $\mathcal{D}(v_n, d-1)$ intersect at v_4.

Since G is pseudo-modular, we learn from this that the three disks have a non-empty intersection. Thus, there exists a vertex x for which (i) $d(v_1, x) \leq 1$, (ii) $d(x, v_3) \leq 1$, and (iii) $d(x, v_n) \leq d-1$,. It follows from (i) and (ii) that can 2-deform $v_1v_2v_3$ to v_1xv_3. Then, by the inductive assumption, we can 2-deform the sub-walk $x \ldots v_n$ to a shortest path from x to v_n. Since by (iii), x already lies on a shortest path from v_1 to v_n, this turns P into a shortest path from v_1 to v_n as desired. ■

We prove next that every pseudomodular graph is stable.

Proposition 3.3.5. *Pseudo-modular graphs are stable.*

For shorthand, a "closed communicating class between s and t" refers to a closed communicating class of the the Markov chain $\mathcal{M}$ defined over the walks from s to t in G ($\Delta = 2$).

Proof We want to show that there is a unique closed communicating class for every s, t.

Assume for contradiction that G is not stable. Then there are two vertices s, t such that there are at least two distinct closed communicating classes from s to t (i.e. of the Markov chain $\mathcal{M}_2(s, t)$). Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two such classes. Note that $\mathcal{C}_1$ and $\mathcal{C}_2$ contain only *shortest paths* from s to t, as shown in Proposition 3.3.4.

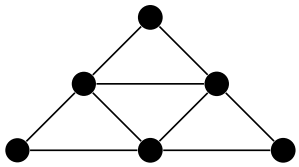
The proof proceeds by induction on the distance between s and t, $d(s, t)$. For the base case, if $d(s, t) \leq 2$ then there clearly must be just one closed communicating class between s and t, a contradiction to C_1 and C_2 being distinct.

To proceed, assume that the statement holds for distances $\leq n-1$, and consider two vertices s and t such that $d(s,t) = n$.

Let $P_1 = v_1 v_2 \dots v_{n+1} \in \mathcal{C}_1$ and $P_2 = u_1 u_2 \dots u_{n+1} \in \mathcal{C}_2$, with $s = v_1 = u_1$ and $t = v_{n+1} = u_{n+1}$, be two shortest paths from s to t. Consider the disks $\mathcal{D}(u_2, 1)$, $\mathcal{D}(v_2, 1)$ and $\mathcal{D}(t, n-2)$. We have that $\mathcal{D}(t, n-2)$ intersects $\mathcal{D}(u_2, 1)$ and $\mathcal{D}(v_2, 1)$ respectively at u_3 and v_3. Furthermore $s \in \mathcal{D}(u_2, 1) \cap \mathcal{D}(v_2, 1)$. Thus there must be a vertex x in the intersection of all disks. For this vertex we have: $d(x, v_2) = 1$, $d(x, u_2) = 1$ and $d(x, t) = n-2$. Since $d(v_2, t) = d(u_2, t) = n-1$, we have that x lies on a shortest path from both u_2 and v_2, to t. Let $P_x = x \dots t$ be some fixed shortest path from x to t. By the inductive assumption we can 2-deform the sub-walks $v_2 \dots v_n$ of P_1 and $u_2 \dots u_n$ of P_2 to the paths $v_2 P_x$ and $u_2 P_x$ respectively. Then we may deform the sub-walk $v_1 v_2 x$ (of the path $v_1 v_2 P_x$) to $v_1 u_2 x$, thus deforming the two paths into the same path. This contradicts that $\mathcal{C}_1$ and $\mathcal{C}_2$ are distinct, so we are done. ■

An example of a graph which is stable but not pseudo-modular is seen in Figure 3.6.

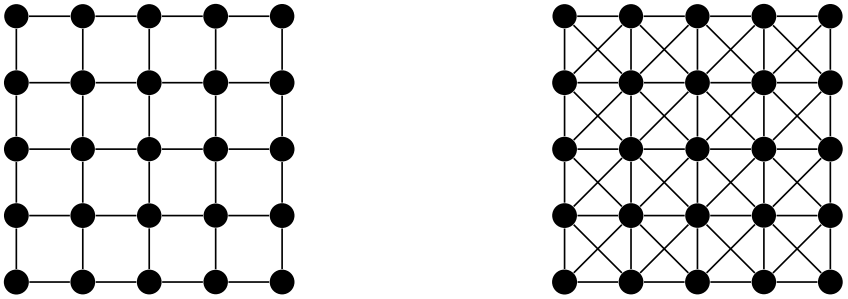
Figure 3.6 Stable, non-pseudo-modular graph



3.3.2 Graph Products

Graph products are a rich topic of study as well as an effective method of creating new topologies from old (see [HIK16; IK00] for an overview). In this section we concern ourselves with two kinds of graph product operations, and show that they preserve graph convergence, and graph stability.

Figure 3.7 Graph products of $\mathcal{P}_5$ with itself



(a) $\mathcal{P}_5 \square \mathcal{P}_5$ **(b)** $\mathcal{P}_5 \boxtimes \mathcal{P}_5$

Definition 3.3.6 (Cartesian product)**.** The Cartesian product of two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, written $G \square H$, is defined to be the graph whose vertices are of the form $u = (x, y)$ where $x \in G$, $y \in H$, and where there is an edge between $(x_1, y_1) \to (x_2, y_2)$ iff $x_1 = x_2$ and $y_1 y_2 \in E_2$, or $y_1 = y_2$ and $x_1 x_2 \in E_1$.

Definition 3.3.7 (Strong product)**.** The strong product of two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, written $G \boxtimes H$, is defined to be the graph whose vertices are of the form $u = (x, y)$ where $x \in G$, $y \in H$, and where there is an edge between $(x_1, y_1) \to (x_2, y_2)$ iff $x_1 = x_2$ and $y_1 y_2 \in E_2$, or $y_1 = y_2$ and $x_1 x_2 \in E_1$, or $x_1 x_2 \in E_1$ and $y_1 y_2 \in E_2$.

Any vertex v of a graph product of G_1 and G_2 can be described as a pair (x, y). The *projection* of v onto G_1 is defined to be x, and its projection onto G_2 is defined to be y. The projection of a walk P onto G_i is the walk over G_i that consists of the projections of the vertices of P onto G_i in order. We will have the notation $d_i(v, u)$ refer to the distance between the projections of the vertices v, u onto G_i. We note that the distance between two vertices v and u over $G_1 \square G_2$ is simply the "taxicab metric" distance [Kra12]; $d_{G_1 \square G_2}(u, v) = d_1(v, u) + d_2(v, u)$. In comparison, it follows from the definition of a strong product that $d_{G_1 \boxtimes G_2}(v, u) = max(d_1(v, u), d_2(v, u))$.

Let $\mathcal{P}_n$ be the path graph on n vertices. Then $\mathcal{P}_n \square \mathcal{P}_n$ is the regular $n \times n$ grid and $\mathcal{P}_n \boxtimes \mathcal{P}_n$ is the grid with diagonals, see Figure 3.7. Therefore this section offers another, different generalization of the known results regarding grid graphs.

Cartesian products

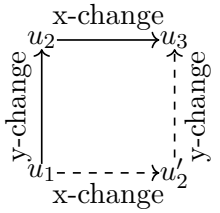
Proposition 3.3.8. *$G_1 \square G_2$ is convergent iff G_1 and G_2 are convergent.*

Proof In one direction, assume $G_1 \square G_2$ is convergent and let, wlog, $P = v_1 v_2 \dots v_n$ be a walk in G_1. Let $u \in G_2$ be an arbitrary vertex. The walk P' whose ith vertex is $v'_i = (v_i, u)$, can be projected onto G_1 and its projection is P. Since P' is 2-deformable to a shortest path from (v_1, u) to (v_n, u), it follows from this that P is 2-deformable to a shortest path from v_1 to v_n (note that any valid 2-deformation of P' over $G_1 \square G_2$ changes only the G_1 coordinate component of the vertices, since 2-deformations never lengthen a path and changing the fixed G_2 component u will do so).

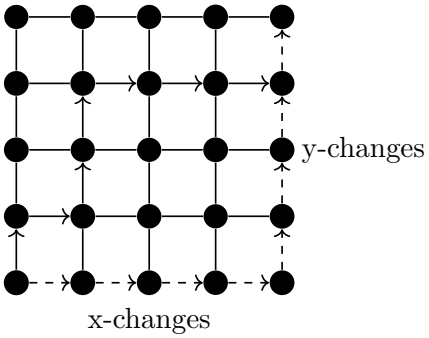
In the other direction, assume G_1 and G_2 are convergent. Let $P = v_1 \dots v_n$ be a walk in $G_1 \square G_2$. We will show it is 2-deformable to a shortest path from v_1 to v_n.

Call an edge in $G_1 \square G_2$ an *x-change* if it affects the G_1 coordinate component and leaves G_2 fixed, else call it a *y-change*. Now let $u_1 u_2 u_3$ be some walk in $G_1 \square G_2$. By the definition of a Cartesian product, if $u_1 u_2$ is a y-change and $u_2 u_3$ is an x-change, we can 2-deform $u_1 u_2 u_3$ into $u_1 u'_2 u_3$ such that $u_1 u'_2$ is an x-change and $u'_2 u_3$ is a y-change. Vice-versa, this is also true. We will call such 2-deformations *swaps*. (See Figure 3.8 for an illustration).

Figure 3.8 The constructions in the proof of Proposition 3.3.8.



(a) A single swap.



(b) 2-deforming the directed path such that all the x-changes occur first

The idea is this: take the walk P and perform swaps on its vertices until all x-changes are consecutive, and all y-changes are consecutive. This results in a walk P' that can be divided into two sub-walks: $P_1' = u_1 \dots u_k$ and $P_2' = u_k \dots u_n$, such that the vertices in P_1' have their G_2-component held constant, and the vertices in P_2' have their G_1-component held constant. We can then use the convergence of G_1 and G_2 to 2-deform these two components to shortest paths, P_1^* and P_2^*. It is simple to see from the definition of a Cartesian product that $P_1^* P_2^*$ must be a shortest path from u_1 to u_n; so we are done. ■

Proposition 3.3.9. *$G_1 \square G_2$ is stable iff G_1 and G_2 are stable.*

Proof The direction where $G_1 \square G_2$ is stable is similar to its counterpart in Proposition 3.3.8. We let $P = v_1 v_2 \dots v_n$ be a walk in G_1 and create a walk P' (as in 3.3.8) whose projection onto G_1 is P. Since from stability it follows that P' is 2-deformable to any shortest path from (v_1, u) to (v_n, u), it follows from this that P is 2-deformable to any shortest path from v_1 to v_n.

In the other direction, assume G_1 and G_2 are stable. Let $P = (x_1, y_1) \to (x_n, y_n)$ be any shortest path in $G_1 \square G_2$. We show that P can always be deformed into a specific shortest path Q, thus showing that $G_1 \square G_2$ is stable (this shows stability via proposition 3.2.9). Let $v_i = (x_i, y_i)$ and let Q_x be a fixed, arbitrary optimal path from $x_1 \to x_n$. Let Q_y be a fixed, arbitrary optimal path from $y_1 \to y_n$. Then Q is defined to be a shortest path of the form $(x_1, y_1)(x_2, y_1) \dots (x_n, y_1)(x_n, y_2) \dots (x_n, y_n)$ (that is, first we go through the path Q_x, holding the y-coordinate fixed, and then through Q_y, holding the x-coordinate fixed).

To deform P to Q, we perform swaps on P as in 3.3.8 to move all the x-changes to the beginning of the path, y-changes to the end, to get a path $P' = P_1' P_2'$. Then similarly to 3.3.8 we can use the stability of G_1 and G_2 to deform the front and end to paths P_1',P_2' to Q_x and Q_y respectively. Thus we deform P to Q as desired. ■

Strong products

For the rest of this section, by $d(v,u)$ we mean the usual distance from v to u over $G_1 \boxtimes G_2$. The fact that $d(v,u) = max(d_1(v,u), d_2(v,u))$ is important and will be used extensively, sometimes implicitly, in the arguments below.

An important thing to note about walks in $G_1 \boxtimes G_2$ is that their x and y components can be 2-deformed independently, so long as the deformation doesn't shorten the walk. More explicitly, consider the walk $P = v_1 \dots v_n$ and denote $v_i = (x_i, y_i)$. Note that if $x_i x_{i+1} x_{i+2}$ is 2-deformable to $x_i x' x_{i+2}$ then $v_i v_{i+1} v_{i+2} = (x_i, y_i)(x_{i+1}, y_{i+1})(x_{i+2}, y_{i+2})$ is 2-deformable to $(x_i, y_i)(x', y_{i+1}), (x_{i+2}, y_{i+2})$. Thus we have changed the projection of P onto G_1 without affecting the projection onto G_2. Equivalently, we can change the projection onto G_2 without changing the projection onto G_1. We will call Δ-deformations that leave either the projection to G_1 or to G_2 unaffected *independent* deformations.

We prove next the following property of strong products on graphs:

Proposition 3.3.10. *$G_1 \boxtimes G_2$ is convergent iff G_1 and G_2 are convergent.*

To aid us we will employ a useful definition.

Definition 3.3.11. Let $P = v_1 \dots v_n$ be some walk of $G_1 \boxtimes G_2$. We define the *x-score* of v_i, $1 < i < n$, to be $d_1(v_{i-1}, v_{i+1}) - d_1(v_i, v_{i+1})$, and the *y-score* to be $d_2(v_{i-1}, v_{i+1}) - d_2(v_i, v_{i+1})$.

Note that the x-score (y-score) receive values only in -1, 0, or 1. The x-score (y-score) of a vertex v_i of the walk $P = v_1 \dots v_n$ is a measure of how much "closer" v_i brings us to v_{i+1} when projected onto G_i, relative to v_{i-1}. If it is positive, then the projection of v_i is closer to v_{i+1} than was that of v_{i-1}.

We start with an observation:

Lemma 3.3.12. *Let $\mathcal{C}$ be a closed communicating class of $G_1 \boxtimes G_2$ (i.e., of the Markov chain $\mathcal{M}_2(s,t)$ for some choice of s and t). Then for any walk $P = v_1 \dots v_n \in \mathcal{C}$:*

1. *If $d_1(v_1, v_3) = 2$ and $d_2(v_1, v_3) \leq 1$, the x-score of every v_i $(1 < i < n)$ is 1*
2. *If $d_2(v_1, v_3) = 2$ and $d_1(v_1, v_3) \leq 1$, the y-score of every v_i $(1 < i < n)$ is 1*
3. *If $d_2(v_1, v_3) = 2$ and $d_1(v_1, v_3) = 2$, then either all v_i have positive x-score, or all v_i have positive y-score*

Since P is Δ-optimal, one of (1), (2) or (3) must hold. Essentially, Lemma 3.3.12 says that one of the projections of a path $P \in \mathcal{C}$ onto either G_1 or G_2 must be Δ-optimal (in G_1 or G_2 respectively), since this is implied by either all of the x-scores or all of the y-scores being positive. The proof idea is to show that whenever this is not the case, P can be "stretched" along either the G_1 or G_2 axes (via Δ-deformation) to a path that is not Δ-optimal, contradicting that $\mathcal{C}$ is a closed communicating class and hence contains only Δ-optimal paths.

Proof For proof of (1), suppose that $d_1(v_1, v_3) = 2$ and $d_2(v_1, v_3) \leq 1$. The proof is by contradiction: Let v_k be the first vertex with non-positive x-score (we assume for contradiction that it exists), meaning that $d_1(v_{k-1}, v_{k+1}) - d_1(v_k, v_{k+1}) \leq 0$, or (alternatively written) $d_1(v_{k-1}, v_{k+1}) \leq d_1(v_k, v_{k+1})$. Recall the definition of 2-optimality, and that every walk in a closed communicating class is 2-optimal. Note that P is 2-optimal, thus, for all i, $d(v_i, v_{i+2}) = 2$ and $d(v_i, v_{i+1}) = 1$. In particular we have that $d(v_k, v_{k+1}) = 1$, which implies that $d_1(v_{k-1}, v_{k+1}) \leq d_1(v_k, v_{k+1}) \leq 1$. Since $d(u, v)$ is the maximum of $d_1(u, v)$ and $d_2(u, v)$, and $d(v_{k-1}, v_{k+1}) = 2$, we must then have that $d_2(v_{k-1}, v_{k+1}) = 2$.

Denote $v_i = (x_i, y_i)$. Using independent deformations, we can 2-deform the vertices of P to have maximized y-scores, as follows: whenever $d(y_i, y_{i+2}) \geq 1$, we 2-deform $y_i y_{i+1} y_{i+2}$ to $y_i y' y_{i+2}$ such that $d(y', y_{i+2}) = d(y_i, y_{i+2}) - 1$ (this is always possible since when y_i and y_{i+2} are not the same vertex there is always a vertex connected to both of them that gets closer to y_{i+2}). We do this without affecting the projection of P on G_1 (and so the x-scores remain unchanged), creating a walk $P' = v_1 v'_2 \dots v'_{n-1} v_n$. Since $d_2(v_1, v_3) \leq 1$, we then have that $d_2(v'_i, v'_{i+2}) \leq 1$ for all i.

Recalling that v_k has non-positive x-score (meaning that so does v'_k), we have that $d_1(v'_{k-1}, v'_{k+1}) \leq 1$ but now also $d_2(v'_{k-1}, v'_{k+1}) \leq 1$, and so $d(v'_{k-1}, v'_{k+1}) \leq 1$, a contradiction to the 2-optimality of all walks in $\mathcal{C}$ and the closure of $\mathcal{C}$ under 2-deformations.

The proof of (2) is the same.

For proof of (3), again by contradiction, let v_k be the first vertex that doesn't have both x- and y-scores positive. Suppose wlog that the y-score is non-positive, meaning we must have $d_2(v_{k-1}, v_{k+1}) \leq d_2(v_k, v_{k+1}) \leq 1$. Due to 2-optimality we must then have $d_1(v_{k-1}, v_{k+1}) = 2$, so we can apply the same argument as (1) to the subwalk $v_{k-1} \dots v_n$ (that is, our new 'v_1' and 'v_3' are v_{k-1} and v_{k+1}), to get that the vertices past v_k have positive x-score as well. ■

We can now move on to the proof of 3.3.10.

Proof The case where $G_1 \boxtimes G_2$ is convergent uses the same idea as propositions 3.3.8 and 3.3.9. Let $P = v_1 v_2 \dots v_n$ be a walk in G_1 and create a walk P' over $G_1 \boxtimes G_2$ (as in 3.3.8) whose projection onto G_1 is P and whose y-coordinate is fixed. We may deform P' into a shortest path, since $G_1 \boxtimes G_2$ is convergent. Any atomic 2-deformation of P' that changes its projection onto G_1 can be translated into a 2-deformation of P by looking only at the changes to the x-coordinates. This implies that P is deformable to a shortest path in G_1.

In the other direction, suppose G_1 and G_2 are convergent. Let $\mathcal{C}$ be a closed communicating class of $G_1 \boxtimes G_2$ and let $P = v_1 \dots v_n$ be a walk in $\mathcal{C}$. We will show P is a shortest path from v_1 to v_n.

Denote $v_i = (x_i, y_i)$. Assume P is not optimal. Then $n - 1 > d(v_1, v_n) \geq max(d_1(v_1, v_n), d_2(v_1, v_n))$, and thus neither of the walks $X = x_1 x_2 \dots x_n$ and $Y = y_1 y_2 \dots y_n$ is optimal. Since G_1 and G_2 are convergent, we can 2-deform these paths

to optimality. Hence, for both X and Y, there is a sequence of atomic 2-deformations that eventually results in a walk of length $n-1$ (i.e. a length one less than their current length). The penultimate element of this sequence will be a walk $u_1u_2\dots u_n$ such that $d(u_{k-1},u_{k+1})\leq 1$ for some k, i.e., a walk that is not 2-optimal (since we cannot delete any vertices from a 2-optimal walk via a single atomic 2-deformation).

Let $X'=x'_1x'_2\dots x'_n$ be a 2-deformation of X such that for some k, $d(x'_{k-1},x'_{k+1})\leq 1$. There must be such a 2-deformation in light of the above. In X', 2-deform the sub-walk $x'_{k-1}x'_kx'_{k+1}$ to $x'_{k-1}x'_{k-1}x'_{k+1}$, calling the new walk X^*. Note that X^* has a vertex with non-positive (zero) x-score.

Let Y' be a 2-deformation of Y such that for some j, $d(y'_{j-1},y'_{j+1})\leq 1$. We again 2-deform the sub-walk $y'_{j-1}y'_jy'_{j+1}$ to $y'_{j-1}y'_{j-1}y'_{j+1}$, resulting in a walk Y^* that has a vertex with non-positive (zero) y-score.

We independently deform the x- and y- components of P into X^* and Y^* respectively. Call the new walk P^*.

According to Lemma 3.3.12, P^* either has all x-scores positive, or all y-scores positive. But we know this to be false due to the way we constructed X^* and Y^*, a contradiction to Lemma 3.3.12. ■

Proposition 3.3.13. *$G_1\boxtimes G_2$ is stable iff G_1 and G_2 are stable.*

Proof The direction where $G_1\boxtimes G_2$ is stable uses precisely the same idea as in Propositions 3.3.8, 3.3.9, and 3.3.10, and is omitted.

Suppose G_1 and G_2 are stable. We will show $G_1\boxtimes G_2$ is stable. To this end we show that every shortest path from $s=(s_x,s_y)$ to $t=(t_x,t_y)$ is deformable to a fixed path Q (see Proposition 3.2.9). We separate the proof into cases.

(1) Assume $d_1(s,t)=d_2(s,t)$. Set $Q=u_1\dots u_n$ to be some arbitrary optimal path from s to t, and set Q_x, Q_y to be the projections over G_1 and G_2 respectively.

Let $P=v_1\dots v_n$ be a path from $s=v_1$ to $t=v_n$. We can assume it is an optimal path due to convergence. Denote by $X=x_1\dots x_n$, $Y=y_1\dots y_n$ the projections of P onto G_1 and G_2. Since $d_1(v_1,v_n)=d_2(v_1,v_n)$, we have that X and Y are both optimal. Thanks to the stability of G_1 and G_2, we can independently 2-deform X to Q_x and Y to Q_y. So any P from s to t is deformable to Q and we are done.

(2) Assume $d_1(s,t)\neq d_2(s,t)$. wlog we will assume that $d_1(s,t)>d_2(s,t)$. Write $n_2=d_2(s,t)+1$ for shorthand. Define Q similar to before, its projections over G_1 and G_2 being as follows: first, Q_x is some arbitrary optimal path from s_x to t_x. Then, the first n_2 vertices of Q_y are some optimal path from s_y to t_y and the rest are t_y repeated ($Q_y=s_y\overset{n_2}{\dots}t_y\ t_y\dots t_y$).

As before let $P=v_1\dots v_n$ be any shortest path from s to t, with the projections X and Y defined as in the previous case. We can deform X to Q_x like before. Deforming Y to Q_y is more delicate.

First we show that given a subwalk $Y(j,k)=y_j\dots y_k$ such that $d_2(y_j,y_k)<k-j$

(that is, the subwalk is sub-optimal), it is possible to deform $Y(j,k)$ such that both the kth and the $(k-1)$th vertices will be equal to y_k.

Since $Y(j,k)$ is sub-optimal and G_2 is stable we can 2-deform $Y(j,k)$ to a walk $Y(j,k)' = y_j y'_{j+1} \dots y'_{k-1} y_k$ such that $d_2(y'_t, y'_{t+2}) = 1$ for some t (the same idea was used in Proposition 3.3.10).

We perform a sequence of 2-deformations on $Y(j,k)'$, explained as follows: first we look at the sub-walk of length 3 $y'_t y'_{t+1} y'_{t+2}$ and 2-deform it to $y'_t y'_{t+2} y'_{t+2}$ (this is possible by the above). We then look at the sub-walk $y'_{t+2} y'_{t+2} y'_{t+3}$ that starts 1 vertex after y'_t (which was $y'_{t+1} y'_{t+2} y'_{t+3}$ before this deformation), and 2-deform it to $y'_{t+2} y'_{t+3} y'_{t+3}$. We continue moving "rightward" in this manner, at every step taking a sub-walk of the form $y'_{t+b} y'_{t+b} y'_{t+b+1}$ and 2-deforming it to $y'_{t+b} y'_{t+b+1} y'_{t+b+1}$:

$$\begin{aligned}
\mathrm{y_j} \dots \mathrm{y'_t y'_{t+1} y'_{t+2}} \dots \mathrm{y'_{k-1} y_k} &\overset{2-def.}{\rightsquigarrow} \mathrm{y_j} \dots \mathrm{y'_t y'_{t+2} y'_{t+2}} \dots \mathrm{y_k} \\
\mathrm{y_j} \dots \mathrm{y'_t y'_{t+2} y'_{t+2} y'_{t+3}} \dots \mathrm{y_k} &\rightsquigarrow \mathrm{y_j} \dots \mathrm{y'_t y'_{t+2} y'_{t+3} y'_{t+3}} \dots \mathrm{y_k} \\
&\vdots \\
\mathrm{y_j} \dots \mathrm{y'_{k-1} y'_{k-1} y'_k} &\rightsquigarrow \mathrm{y_j} \dots \mathrm{y'_{k-1} y_k y_k}
\end{aligned}$$

This eventually gives the desired deformation: a walk that ends with $y_k y_k$.

Let $o \geq n_2$ be the earliest index of the vertex t_y in the walk Y. We deform $Y(1,o)$ a finite number of times using the above idea, to duplicate t_y to the index n_2, resulting in a walk Y'. The last (nth) vertex of Y' is t_y, and so is the n_2th vertex, so the subwalk $Y'(n_2, n)$ is sub-optimal. Thus we may repeatedly apply the above 2-deformations to duplicate t_y across the rest of the walk. This leaves us with a 2-deformation of Y, Y'', such that the first n_2 vertices are an optimal path from s_y to t_y, and the rest of the vertices are t_y. Finally we 2-deform the first n_2 vertices (using the stability of G_2) to equal those of Q_y, and we are done. ■

An interesting corollary of the above propositions is the following:

Corollary 3.1. *$G_1 \boxtimes G_2$ is stable (resp., convergent) iff $G_1 \square G_2$ is stable (resp., convergent)*

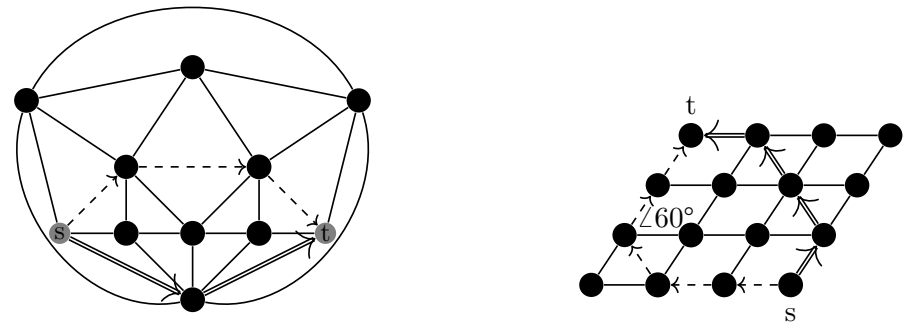
In other words, for questions of stability and convergence of probabilistic pursuit, there is no difference between the topology induced by the "taxicab" distance $d((x_1,y_1),(x_2,y_2)) = d_1(x_1,x_2) + d_2(y_1,y_2)$ and the topology induced by the distance $d((x_1,x_2),(y_1,y_2)) = max(d_1(x_1,y_1), d_2(x_2,y_2))$.

3.3.3 Planar and Chordal Graphs

It is surprisingly difficult to pinpoint the effect of the underlying graph being *planar* on chain pursuit. Certainly, not every planar graph is convergent - a simple counterexample is a cycle of length 5 and above. But one might expect that planar graphs with

high connectivity, or other good "regularity" properties (for example, low complexity of the planar graph's faces), would be convergent, if not stable. The counterexamples in Figure 3.9 highlight the difficulty of pinpointing such properties. Figure 3.9, (a) shows a maximal planar graph that is not convergent. Figure 3.9, (b) shows a matchstick graph (a planar graph that can be drawn on the plane with all edge lengths being 1) whose faces are all triangular or square. The dashed edges show a suboptimal recurrent path from s to t; the double lines show the optimal path.

Figure 3.9 Some planar, non-convergent graphs.



(a) **(b)**

An outerplanar graph is a graph that has a planar drawing for which all vertices belong to the outer face of the drawing. A maximal outerplanar graph is an outerplanar graph to which we can add no edges. A positive result for planar graphs is the following:

Proposition 3.3.14. *Every maximal outerplanar graph is convergent.*

In fact, this proposition stems from a more general observation. A fairly well-known class of graphs are the *chordal graphs* (see [Gol04b; Dir61] for background). Chordal graphs can be defined in two equivalent ways.

Definition 3.3.15. A *simplicial vertex* is a vertex whose neighbors form a complete graph [Gol04b].

Definition 3.3.16. The following are equivalent characterizations of chordal graphs:

1. All cycles of four or more vertices of G have a chord (an edge that is not part of the cycle but connects two vertices of the cycle).

2. G has a *perfect elimination ordering*: an ordering of the vertices of the graph such that, for each vertex v, v is a *simplicial vertex* of the graph induced by v and the vertices that occur after v in the order.

Definition (1) hints at a "regularity" property of the kind we are looking for that is possessed by chordal graphs.

It is well-known and simple to prove that every maximal outerplanar graph is chordal. To see this, note that the regions in the interior of a maximal outerplanar

graph form a tree (if there was a cycle, it would necessarily surround some vertex of the graph, which contradicts outerplanarity). As such any region of the outerplanar graph corresponding to a leaf of that tree must have a vertex of degree 2. It is simple to see that this vertex is simplicial, and that after its removal the graph remains maximal outerplanar. Thus we found a perfect elimination ordering, and every such graph must be chordal. (See the famous "ear clipping" algorithm [EET93]).

Thus, proposition 3.3.14 is a consequence of the following:

Proposition 3.3.17. *Chordal graphs are convergent.*

Proof The proof is by induction. We assume every chordal graph of size $n-1$ is convergent, and prove that this yields the result for graphs of size n. In the base case, it is simple to verify that any graph (chordal or not) with $n \leq 4$ vertices is convergent.

Let G be a chordal graph of size n and let v be a simplicial vertex of G. Consider a walk $P = u_1 \dots u_m$ in G. We show that P can be 2-deformed into a shortest path. We separate our proof into three cases:

(1) If v does not occur as a vertex in P, then we can 2-deform P into a shortest path just as we would working in $G - v$ (which is chordal and therefore convergent by our inductive assumption). (Note that since v is simplicial, it does not occur in any shortest path from u_1 to u_m).

(2) If v occurs in P as a vertex u_i, $1 < i < m$, then since it is simplicial we have $d(u_{i-1}, u_{i+1}) = 1$, thus we can 2-deform $u_{i-1}u_iu_{i+1}$ to $u_{i-1}u_{i+1}$ and remove v from P. Thus we are reduced to either case (1) or case (3).

(3) Either $u_1 = v$ or $u_m = v$ and v occurs nowhere else in P. Here we require another separation to cases:

In the first case, both $u_1 = v$ and $u_m = v$. Using the convergence of $G - v$ we can 2-deform $u_2 \dots u_{m-1}$ to u_2u_{m-1} (since u_2, u_{m-1} are neighbors of v and therefore $d(u_2, u_{m-1}) \leq 1$). This leaves us with the walk $P' = u_1u_2u_{m-1}u_m$, and we have $d(u_1, u_{m-1}) = 1$, so we may remove u_2 from it via 2-deformation. Finally since $d(u_1, u_m) = 0$ we can remove u_{m-1}, leaving us with $P'' = u_1u_m$, an optimal path.

In the other case, we assume wlog that only $u_1 = v$ (the case where $u_m = v$ is symmetrical). By the inductive assumption, and since $G - v$ is chordal, we can 2-deform the subwalk $u_2u_3 \dots u_m$ to a shortest path. We will assume wlog that it already is. Since $\Delta = 2$, we can assume that $|P| = m > 3$, otherwise the proof is trivial.

Suppose that in spite of $u_2u_3 \to \dots u_m$ being optimal, the walk P is not a shortest path from u_1 to u_m. Recall the definition of discrepancy vertices. Since it is not optimal, P must contain a pair of discrepancy vertices. Since the subwalk $u_2 \dots u_m$ is optimal, this must be a pair of the form u_1, u_k for some k (as optimal subwalks contain no discrepancy vertices).

For simplicity, will assume that $k = m$, and deal with the case where $k \neq m$ at the end. That is, we assume that u_1 and u_m are the discrepancy vertices.

Let $P^* = v_1 \dots v_l$ be a shortest path from $v_1 = u_1 = v$ to $v_l = u_m$. Since v is a simplicial vertex and u_2 is a neighbor of v, we have that $d(u_2, u_m) \leq d(v, u_m)$. Note that since the sub-walk $u_2 u_3 \dots u_m$ is optimal, and goes through $m-2$ edges, we have that $m-2 = d(u_2, u_m)$. In turn this implies that $|P^*| = d(v, u_m) + 1 \geq d(u_2, u_m) + 1 = m - 1 \geq 3$.

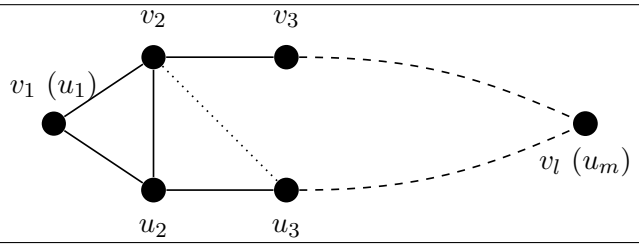
Since u_1 and u_m are discrepancy vertices, the paths P^* and P contain no shared vertices except at the endpoints (see Lemma 3.3.2). Thus the vertices of $P \cup P^*$ form a cycle. Since $|P^*|, |P| \geq 3$, this cycle contains both u_3 and v_3 as vertices.

The subgraph induced by the vertices of the cycle $H = P \cup P^*$ is a sub-graph of a chordal graph, thus it is chordal. Since $v (= u_1)$ is simplicial, the graph $H - v$ is also chordal. Note that the edge $u_2 v_2$ exists (since u_2 and v_2 are neighbors of v), and is an edge of the cycle $v_2 \dots v_l \dots u_2$ in $H - v$. Hence, there must be a cycle in $H - v$ with minimal number of vertices containing the edge $u_2 v_2$. This cycle must be of size 3, since $H - v$ is chordal. We note the following facts:

1. This cycle is either of the form $u_2 v_q v_2$ or $u_2 u_q v_2$, for $q > 2$.
2. If it is of the form $u_2 v_q v_2$, then $u_1 u_2 v_q v_{q+1} \dots v_l$ is a shortest path from u_1 to $v_l = u_m$ (since $u_1 v_2 v_3 \dots v_l$ is a shortest path and $q > 2$). This contradicts the fact that u_2 must not belong to such a path (since u_1 and u_m are discrepancy vertices). Therefore this is an impossibility.
3. If it is of the form $u_2 u_q v_2$, and $q > 3$, then there is an edge from u_2 to u_q. This is a contradiction to the fact that $u_2 u_3 \dots u_m$ is a shortest path. Therefore, we must have that $q = 3$.

Thus we see that this cycle must be precisely the cycle $u_2 u_3 v_2$. Therefore, the edge $v_2 u_3$ must exist in G.

Figure 3.10 The induced subgraph $P \cup P^*$



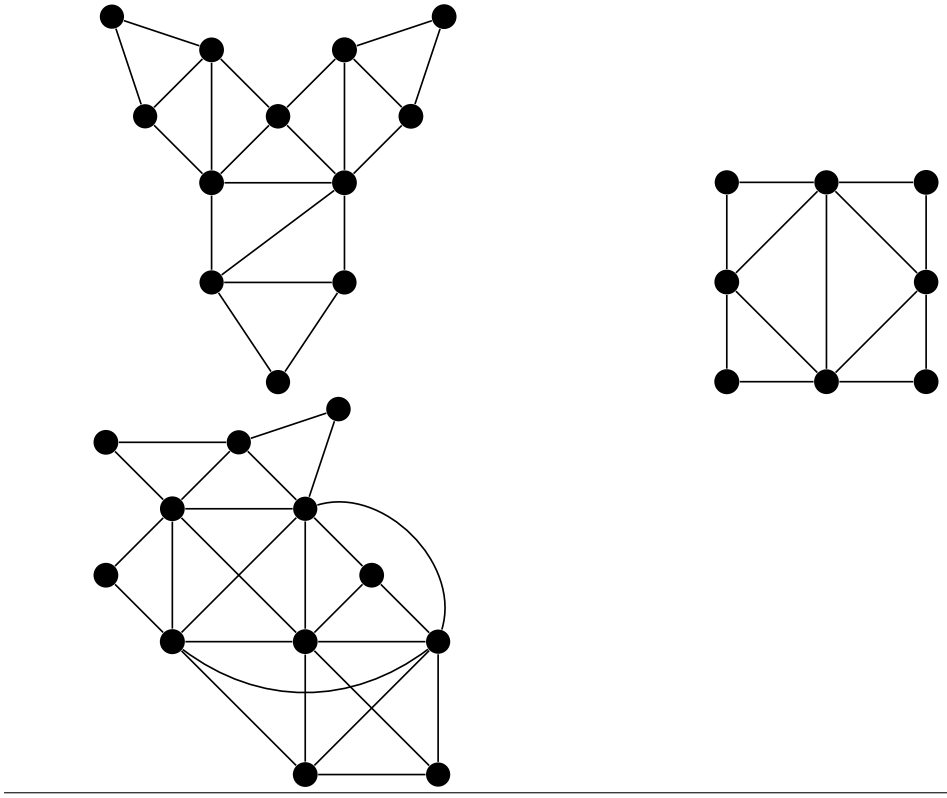
Since $v_2 u_3$ exists, we can 2-deform $u_1 u_2 u_3$ to $u_1 v_2 u_3$, then 2-deform $v_2 u_3 \dots u_m$ to a shortest path from v_2 to u_m (using the inductive assumption), deforming P into an optimal path. Thus we successfully 2-deformed P into an optimal path, and we are done.

If $k \neq m$ we first restrict ourselves to looking at the sub-walk $u_1 \dots u_k$ of P and apply the argument above to 2-deform it to a shortest path. This has the effect of shortening P. If P is not a shortest path as a result of this, we can freely re-apply the

same argument a finite number of times to deform P to a shortest path. Specifically, after taking care of v_1, v_k we look for the next pair of discrepancy vertices and apply the argument to them, each time shortening the length of P by at least 1. This can only be done a finite number of times (as P is finite), and at the end of this process we will have deformed P to a shortest path. ■

Note that in general chordal graphs are not necessarily planar.

Figure 3.11 The top row shows two maximal outerplanar graphs; the bottom row a non-planar chordal graph

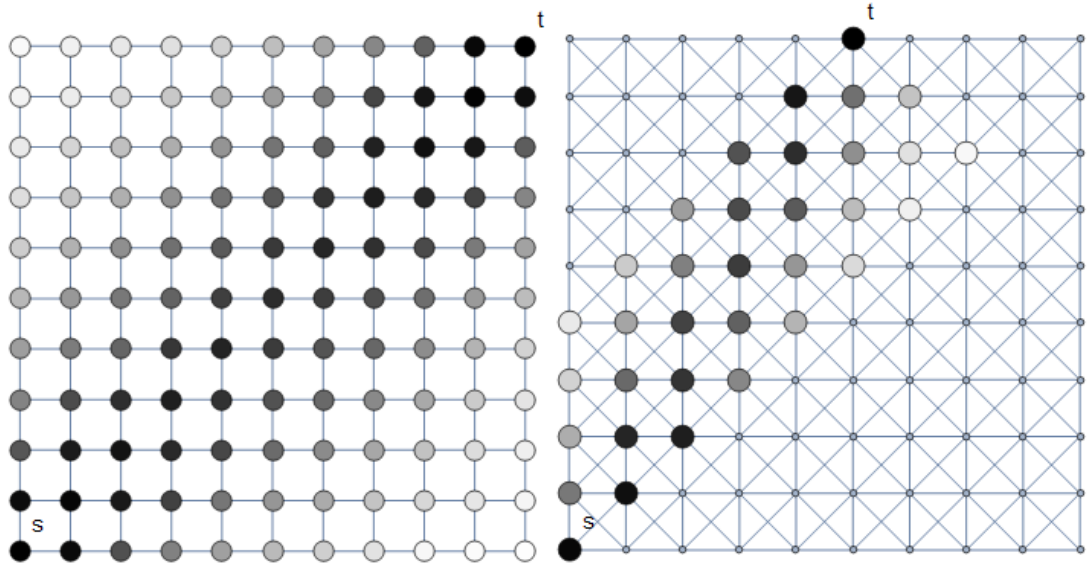


3.4 The Uniform Stationary Distribution

In the previous sections we discussed *stable graphs*, graphs for which the pursuit from s to t converges to a unique distribution over all shortest paths. In other cases, while chain pursuit always converges to a distribution over some set of walks, this distribution is not uniquely determined and depends on the initial walk $P(A_0)$ as well as the randomness of each ant's path choices.

The purpose of this section is to prove a general fact about these distributions; namely, that probabilistic chain pursuit will always converge to the *uniform* distribution over a set of walks in one of its closed communicating classes. When restricting the discussion to stable graphs, this says that the pursuit will always converge to the uniform distribution over all shortest paths from s to t.

Figure 3.12 A simulation of chain pursuit on different graph environments. Each vertex is shaded according to the relative frequency at which an agent A_i was located on it. A corollary of this section is that for a vertex v, this frequency converges to precisely the number of walks in $\mathcal{C}$ (the closed communicating class at which the pursuit stabilized) that pass through v, divided by $|\mathcal{C}|$. In the graphs pictured, the most frequented vertices are those closest to the straight line from s to t.



Proposition 3.4.1. *Let $\mathcal{C}$ be a closed communicating class of $\mathcal{M}_\Delta(s,t)$. The unique stationary distribution of $\mathcal{M}$ restricted to $\mathcal{C}$ is the uniform distribution.*

In particular we have:

Proposition 3.4.2. *Let G be a stable graph. Then the unique stationary distribution of $\mathcal{M}_\Delta(s,t)$ (for any choice of Δ) is the uniform distribution over all shortest paths from s to t.*

We note that for the purposes of our proof, unlike the previous sections, we cannot restrict the parameter Δ here to the value '2', and our argument must hold for any value Δ greater than 1. This is because the transition probabilities of $\mathcal{M}_\Delta(s,t)$ depend on the choice of Δ.

The technique we will use to prove Proposition 3.4.1 is a generalization of the *statistical exchangeability* technique shown in the Preliminaries chapter. To see this, let us first give a proof of Proposition 3.4.1 assuming G is the $n \times n$ grid graph. The proof of this special case is due to [BMW97].

Proposition 3.4.3. *(Special case of Proposition 3.4.1 for grid graphs.) Let G be the $n \times n$ grid graph and let $s = (0,0), t = (n,n)$. Then the unique stationary distribution of $\mathcal{M}_\Delta(s,t)$ is the uniform distribution over all shortest paths from $(0,0)$ to (n,n).*

Proof We will assume that $P(A_0)$ is uniformly distributed over all paths in $\mathcal{C}$ and show that the induced distribution of $P(A_1)$ is the uniform distribution. This shows that the uniform distribution is stationary. Since by Proposition 3.3.9 G is stable, we know it has a unique stationary distribution, completing the proof.

The number of different shortest paths from $(0,0)$ to (n,n) is $S = \binom{2n}{n}$. The ith vertex of $P(A_0)$ is (x,y) with probability

$$\frac{1}{S}\binom{i}{x}\binom{2n-i}{n-x} \tag{3.1}$$

This is the hypergeometric distribution, which governs the number of white balls (x) in a sample of i balls from an urn that initially has a white and b black balls. We can generate a uniformly random path by drawing balls sequentially at random from this urn. Recalling our pursuit rule, we can obtain the distribution of A_1's path by considering the following process with two urns called Urn 0 and Urn 1:

Urn 0 initially contains n white and n black balls. At time unit $t \leq 2n$ a ball is drawn from Urn 0 uniformly at random and placed inside Urn 1 - this ball represents the t-th vertex of $P(A_0)$. Once Δ balls are in Urn 1 (i.e., at time $t = \Delta$), at each time unit we begin also drawing balls uniformly at random from Urn 1, representing the vertices of $P(A_1)$.

The distribution of $P(A_1)$ is given by the distribution of the balls drawn from Urn 1. Here is where *exchangeability* comes in: let us label the balls $1..2n$, disregarding their color, and let X_i equal the ith ball that is drawn from U_1. Then $P(A_1)$ is determined by the sequence $U = X_1, X_2 \ldots X_{2n}$. Note that by symmetry, $X_1, X_2 \ldots X_{2n}$ is an exchangeable sequence of random variables. Hence U is equally likely to be any of the $2n!$ possible permutations, implying that $P(A_1)$ is uniformly distributed. ■

Our proof of Proposition 3.4.1 will generalize that of Proposition 3.4.3. Let $\mathcal{C}$ be a closed communicating class of $\mathcal{M}_\Delta(s,t)$, and consider the Markov chain defined by deleting every walk not in $\mathcal{C}$ from $\mathcal{M}$. This restricted Markov chain is finite, irreducible and aperiodic (as every walk taken by an ant has a non-zero probability of immediately recurring for the next ant) and so has a unique stationary distribution [LPW09]. We will assume that $P(A_0)$ is uniformly distributed over all paths in $\mathcal{C}$ and show that the induced distribution of $P(A_1)$ is the uniform distribution. This is equivalent to showing that the uniform distribution is the unique stationary distribution of the restricted Markov chain.

Definition 3.4.4. Let $X = x_1 \ldots x_n \in \mathcal{C}$ be a walk in G. An *(i,j)-shuffle* of X, written $S^i_j(X)$, is a random variable resulting from the replacement of the sub-walk $x_i \ldots x_j$ of X with a shortest path from x_i to x_j chosen uniformly at random from all such paths. (We define $S^i_i(X) = X$ and $S^i_{n+k}(X) = S^i_n(X)$ for $k \geq 1$).

Consider the pursuit of A_0 by the ant A_1. The pursuit rule states that A_1 moves, at time i, to a vertex determined by choosing at random one of the shortest paths from

A_1 to A_0 and stepping on the first vertex of that path. As we will see, this is equivalent to sequentially performing $(i, i+\Delta)$-shuffles of $P(A_0)$ starting from $i = 1$ up to $i = n$.

We define $\eta(u, v)$ to be the number of shortest paths from u to v in G, and we define $\eta(v, v) = 1$. In a slight abuse of notation, we will also let $\eta(u, v)$ be the *set* of all such shortest paths, trusting that the intent will be clear from the context. We further define $\eta(u_1u_2, v)$ to be the set of all shortest paths from u_1 to v, whose first edge is u_1u_2 (i.e. paths with $d(u_1, v)$ edges whose first edge is u_1u_2). We note that $\eta(u_1u_2, v)$ might equal 0 for some choices of u_1u_2 and v.

For simplicity, we define $x_{n+k} = x_n$ for $k \geq 1$, in all applications below.

For the next several lemmas, let $U = u_1 \dots u_n$ be an arbitrary, fixed walk in $\mathcal{C}$. Write $p_U(X)$ for the probability $Prob[P(A_1) = X | P(A_0) = U]$. We start with the following lemma:

Lemma 3.4.5. *Let $X = x_1 \dots x_n$ be a walk in $\mathcal{C}$ such that $p_U(X) > 0$. Then we have:*

$$p_U(X) = \prod_{i=1}^{n} \frac{\eta(x_i x_{i+1}, u_{i+\Delta})}{\eta(x_i, u_{i+\Delta})}$$

Proof In order for A_1 to have $P(A_1) = X$, it must, at time $i+1$, choose the vertex x_{i+1}, having already chosen the vertex x_i at time i. At this time A_0 will be on the vertex $u_{i+\Delta}$, which by Δ-optimality is at distance Δ from x_i. By the pursuit rule, it follows that moving to the vertex x_{i+1} happens with probability

$$\frac{\eta(x_i x_{i+1}, u_{i+\Delta})}{\eta(x_i, u_{i+\Delta})}$$

The formula follows from multiplication of all these probabilities. ■

We define the following stochastic process: let $U^1 = S^1_{1+\Delta}(U)$, and $U^i = S^i_{i+\Delta}(U^{i-1})$. Define $\widetilde{p}_U(X)$ to be the probability that U^n will equal X. We will show:

Lemma 3.4.6. *For all $X \in \mathcal{C}$, $p_U(X) = \widetilde{p}_U(X)$.*

Proof The ith shuffle permanently determines the $(i+1)$th vertex. We have $u_1 = x_1$ and $u_n = x_n$. In order for u_2 to be changed into x_2 we must have that the second vertex of $S^1_{1+\Delta}(U)$ is x_2, which by the definition of η happens with probability

$$\frac{\eta(x_1 x_2, u_{1+\Delta})}{\eta(x_1, u_{1+\Delta})}$$

Inductively, we again arrive at the formula:

$$\widetilde{p}_U(X) = \prod_{i=1}^{n} \frac{\eta(x_i x_{i+1}, u_{i+\Delta})}{\eta(x_i, u_{i+\Delta})}$$

Which shows that $p_U(X) = \widetilde{p}_U(X)$. ■

From the proof of 3.4.6 we see that the probability distribution of U^n is equivalent to that of $P(A_1)$ (conditioned on $P(A_0) = U$).

Lemma 3.4.7. *Let $U = u_1 \dots u_n$ be chosen uniformly at random from the walks in $\mathcal{C}$. For all $X = x_1 \dots x_n \in \mathcal{C}$, $Prob[U^n = X] = \frac{1}{|\mathcal{C}|}$.*

Proof According to our assumption, for any $X \in \mathcal{C}$, we have $Prob[U = X] = \frac{1}{|\mathcal{C}|}$. Then for any $1 \leq i < j \leq i + \Delta$ we have that

$$Prob[S_j^i(U) = X] = \frac{\eta(u_i, u_j)}{|\mathcal{C}|} \cdot \frac{1}{\eta(u_i, u_j)} = \frac{1}{|\mathcal{C}|}$$

(The computation relies on the fact that the subwalk from u_i to u_j in U must be Δ-deformable to any path in $\eta(u_i, u_j)$, thus by closure, the result of any such deformation must be in $\mathcal{C}$).

We see that the distribution of $S_j^i(U)$ is the same as U. Since U^n is just a composition of a finite number of such shuffles, we have that $Prob[U^n = X] = \frac{1}{|\mathcal{C}|}$. ■

The above lemma shows that the distribution of $P(A_1)$ is the uniform distribution, completing the proof of Proposition 3.4.1. ■

3.5 Discussion

We studied the behavior of "ants" that form an idealized "ant trail" through probabilistic chain pursuits over an arbitrary graph environment G. Our pursuit rule is a natural generalization of the pursuit rule of [BMW97]. Unlike the original simpler scenario where pursuit was restricted to a grid graph, in general graph environments, chain pursuit does not necessarily converge to the uniform distribution over all shortest paths as time tends to infinity. In fact, depending on the random choices of the ants, chain pursuit may stabilize in several different ways even over the same graph and same source and destination vertices. We therefore investigated conditions under which convergence and stability do occur. In doing so we extended the results of [BMW97] on the grid in multiple ways: by showing several classes of graphs that are convergent to shortest paths and are stable, and by showing that the limiting distribution of walks in probabilistic pursuit is always uniform. From the graph classes investigated in Section 3.3, two of these, pseudo-modular graphs and graph products, include the special case where the underlying graph is a grid.

Three immediate extensions of this work can readily be considered. The first is a classification of stable and convergent graphs with respect to a parameter Δ greater than 2, i.e., graphs where convergence to the shortest path (or, additionally, to a unique stationary distribution) is guaranteed only for choice of Δ greater than 2. The second is further investigation of types of planar graphs that are stable or convergent. The third is an analysis of the computational complexity of the problem of deciding whether a given graph is convergent or stable.

Chapter 4

Natural Algorithms II: A Locust-inspired Model of Collective Marching on Rings

Continuing our investigation of natural algorithms, in this chapter we study a model of collective motion for agents in ringlike environments. The model's dynamics is inspired by known laboratory experiments on the dynamics of locust swarms. In these experiments, locusts placed at arbitrary locations and initial orientations on a ring-shaped arena are observed to eventually all march in the same direction. In this chapter we ask whether, and how fast, a similar phenomenon occurs in a stochastic swarm of simple locust-inspired agents. The agents are randomly initiated as marching either clockwise or counterclockwise on a discretized, wide ring-shaped region, which we subdivide into k concentric tracks of length n. Collisions cause agents to change their direction of motion. To avoid this, agents may decide to switch tracks so as to merge with platoons of agents marching in the same direction of motion as themselves.

We shall mostly follow the works [AAB21; AAB22]. The analysis combines three techniques from our toolbox: coupling (e.g., in Lemma 4.4.3), potential functions (Lemma 4.4.15), and stationary distributions (Lemma 4.4.17).

4.1 Introduction

Birds, locusts, human crowds and swarm-robotic systems exhibit interesting collective motion patterns. The underlying autonomous agent behaviours from which these patterns emerge have attracted a great deal of academic interest over the last several decades [APB18; WAYB08; AA15; FK10; GCJT13; AYWB08]. In particular, the formal analysis of models of swarm dynamics has led to varied and deep mathematical results [Bru93; CGG$^+$08; CVV99; RMB19]. Rigorous mathematical results are necessary for understanding swarms and for designing predictable and provably effective swarm-robotic systems. However, multi-agent swarms have a uniquely complex and

"mesoscopic" nature [Cha18], and relatively few standard techniques for the analysis of such systems have been established. Consequently, the analysis of new models of swarm dynamics is important for advancing our understanding of the subject.

In this work, we study the dynamics of "locust-like" agents moving on a discrete ringlike surface. The model we study is inspired by the following well-documented experiment [AAA16]: place many locusts on a ringlike arena at random positions and orientations. They start to move around and bump into the arena's walls and into each other, and as they do so, remarkably, over time, they begin to collectively march in the same direction–either clockwise or counterclockwise (see Figure 4.1). Inspired by observing these experiments, we asked the following question: what are simple and reasonable myopic rules of behaviour that might lead to this phenomenon? Our goal is to study this question from an *algorithmic* perspective, by considering a swarm of autonomous and identical discretized mobile agents that act according to a local algorithm. The precise mechanisms underlying locusts' behaviours are very complex and subject to intense ongoing research, e.g. [AA15; AAA16; AOL$^+$14; KSA$^+$21; KAGA19]. Consequently, as with much of the literature on swarm dynamics [Cha12; Bru93; AB19a], our goal is not to study an exact mathematical model of locusts in particular, but to study the kinds of algorithmic local interactions that lead to collective marching and related phenomena. The resulting model is idealized and simple to describe, but the patterns of motion that emerge while the locusts progress towards a "stabilized" state of collective marching are surprisingly complex.

Figure 4.1 Image from locust experiments, courtesy of Amir Ayali. The collective clockwise marching of locusts in a ring arena is shown. Locusts were initiated at random positions and orientations in the arena, but converged to clockwise marching over time.

The starting point for this work is the following postulated "rationalization" of what a locust-like agent wants to do: it wants to keep moving in the same direction of motion (clockwise or counterclockwise) for as long as possible. We can therefore consider a model of locust-like agents that never change their heading unless they collide, heads-on, with agents marching in the opposite direction, and are forced to do

so due to the pressure which is exerted on them. When possible, these agents prefer to *bypass* agents that are headed towards them, rather than collide with those agents. This is done by changing lanes: moving in an orthogonal manner between concentric narrow *tracks* which partition the ringlike arena. The formal description of this "rationalized" model is given in Section 4.3, and will be our subject of study.

Contribution.

We describe and study a stochastic model of locust-inspired agents in a 2D discretized ringlike arena which is subdivided into k tracks each consisting of n locations. We show that our agents eventually reach a "local consensus" about the direction of marching, meaning that all agents on the same track will march in the same direction. We give asymptotic bounds for the time this takes based on the number of agents and the physical dimensions of the arena. Due to the idealized deterministic nature of our model, a global consensus where *all* locusts walk in the same direction is not guaranteed, since locusts in different tracks might never meet. However, we show that, when a small probability of "erratic", random behaviour is added to the model, such a global consensus must occur. We verify our claims via simulations and make further empirical observations that may inspire future investigations into the model.

Despite being simple to describe, analyzing the model proved tricky in several respects. Our analysis strategy is to show that the model oscillates between two phases: one in which it is "chaotic," and locusts are moving about without a discernible pattern, and one in which it is "orderly," and all locusts are stuck in dense deadlock situations where collisions are frequent. We derive our asymptotic bounds from studying orderly phases while bounding the amount of time the locusts can spend in chaotic phases.

Previous works in the literature (e.g., [BSC+06; YEE+09]) have explained collective marching by appealing to a principle of local averaging, wherein each agent attempts to average its direction of motion with its neighbors'. It is interesting to note that our model attains collective marching from nearly the opposite set of assumptions: our agents' primary motivation is to *avoid* changing their direction of motion, and any change to it is thus the result of an unavoidable conflict. We refer the reader to the Related Work section below for further discussion.

4.2 Related work

The locust experiments inspiring our work are discussed in [AA15; AAA16; AOL+14; KSA+21; BSC+06; YEE+09]. The phenomenon was originally studied by Buhl et al. [BSC+06]. They show that above a certain critical density, a rapid transition occurs from disordered movements of locust nymphs to highly aligned collective motion. Buhl et al., and subsequently Yates et al. [YEE+09] hypothesize that a main cause of this behaviour is the locusts' tendency to change their direction to align with neighbors

within a local interaction range (a common modelling assumption in multi-agent dynamics [CVV99]) and that individual behavior does not change in relation to group density. In this work we show how collective marching might emerge from almost the opposite set of assumptions: the locust-like agents we describe try to *avoid* changing their direction of motion for as long as possible, going as far as actively avoiding locusts that are headed in the opposite direction, but consensus eventually occurs as a result of unavoidable conflicts where locusts bump into each other. The assumption that locusts want to maintain their direction of motion is critical for enabling collective marching in our model, since it characterizes the stable states of the system. Bazazi et al. [BBH$^+$08] hypothesize collective marching occurs due to a model of cannibalistic pursuit wherein locusts attempt to pursue locusts in front of them and evade locusts behind them to bite and avoid being bitten. Our model includes an element of evasion, too, but it is motivated by the locusts' desire to avoid changing their direction of motion. All previous models assume local interactions between locusts, i.e., locusts are only affected by neighboring locusts. Interactions in our model consist of conflicts between adjacent locusts and track-changes that occur as a result of trying to avoid said conflicts. Conflicts are by definition local. Track-changing rules can be assumed either local or global and our analysis applies in both cases.

Notably, in [BSC$^+$06; YEE$^+$09] it is observed that at intermediate densities, swarms of locusts exhibit periodic directional switching, and at low densities the directions of motion are random. Our model does not replicate these phenomena–we show that our locust-like agents converge to local consensus at every density (or global consensus, assuming noise). Interestingly, we note that if we assume each locust has a small probability $r > 0$ of randomly flipping their heading at the beginning of a time step, such directional switching becomes possible. The probability of directional switching under such a postulate is inversely proportional to the density, thus likelier at low and intermediate densities than at high densities. We emphasize, however, that unlike works such as Buhl et al., replicating all features of locust swarms is not the goal of this work. Whereas the works we discussed seek to model actual locusts, our work can be characterized as trying to find a minimalistic locust-*inspired* set of assumptions that provably attains collective marching and to study it analytically for the sake of deepening our understanding of multi-agent systems.

More generally, the mathematical modelling of the collective motion of natural organisms such as birds, locusts and ants, and the convergence of such systems of agents to stable formations, has been discussed in numerous works including [CVV99; GCJT13; RMB19; SA15]. The most relevant to us among these are works within the field of *natural algorithms*, which asserts that the behaviour of natural organisms can be understood using concepts from the theory of robotics and computer science [Cha12; AB19a; Cha18], such as complexity analysis, look-compute-move phases, and decision-making based on discrete internal states. Natural algorithms open up interplay between biology and computer science, allowing us to study nature via the language of algo-

rithms and vice-versa, allowing us to translate principles, algorithms and mechanisms gleaned from nature to the design of systems that are meant to service or interact with humans, such as autonomous vehicles and warehouse robots.

The central focus of this work regards consensus: do the agents eventually converge to the same direction of motion, and how long does it take? These questions bear mathematical and conceptual resemblance to questions in the field of opinion dynamics [APP12; YOA+13; XWX11]. If the agents' direction of motion (clockwise or counterclockwise) is considered an "opinion," and the agents' interactions that cause changes in the direction of motion are considered social pressure, we can ask how long does it take for the agents to arrive at a consensus of opinions. Building on this analogy, we note that when there are no empty locations in the environment, our agent model is distinctly similar to the *voter model* on a ring network with two opinions. The voter model is a classical model in opinion dynamics explored in numerous works (we refer the reader to the survey [DZK+18]).

The comparison to the voter model breaks when we introduce empty locations and multiple ringlike tracks, at which point we must take into account the agents' dynamically changing positions. Unlike the voter model, where only an agent's static neighborhood can influence its opinion, in our model an agent's current location determines which agents can influence it. Several works have explored models of opinion dynamics in a ring environment where the agents' physical location is taken into account [CB17; HMW16]. Our model is distinct from these in several respects: first, in our model, an agent's internal state–its direction of motion–plays an active part in the algorithm that determines which locations an agent may move to. Second, we partition our ring topology into several narrow rings ("tracks") that agents may switch between, and an agent's decision to switch tracks is influenced by the presence of platoons of agents moving in its direction in the track that it wants to switch to. In other words, we model agents that actively attempt to "swarm" together with agents moving in their direction of motion. We believe our work is unique in that we study, in a single model, both how an agent's physical location affects its opinion (via conflicts with nearby agents), and how an agent's opinion affects its physical location (via the desire to swarm with agents of the same opinion or equivalently, evade those of a different opinion).

Protocols for achieving consensus about a value, location or the collective direction of motion have also been investigated in swarm robotics and distributed algorithms [BMB17; Cor08; MB18; OFM07]. The purpose of these protocols is typically to be as efficient as possible in terms of parameters such as time, computational load, and distance travelled. However, in this work, we are not searching for a protocol that is designed to efficiently bring about consensus; we are investigating a protocol that is inspired by natural phenomena and want to see *whether* it leads to consensus and how long this process is expected to take.

Broadly speaking, some mathematical similarities may be drawn between our model and interacting particle systems such as the simple exclusion process, which have been

used to understand biological transport and traffic phenomena [CMZ11; AB20]. Such particle systems have been studied on rings [KK10]. In these discrete models, as in our model, agents possess a physical dimension, which constrains the locations they might move to in their environment. These are not typically multi-agent models where agents have an internal state (such as a persistent direction of motion), but rather models of particle motion and diffusion, and the research focus is quite different; the main point of similarity to our model is in the way that a given discrete location can only be occupied by a single agent, and in the random occurrence of "traffic shocks" wherein agents line up one after the other and are prevented from moving for a long time.

4.3 Model and definitions

We postulate a locust-inspired model of marching in a wide 2D ringlike arena which is discretized into k narrow concentric rings each consisting of n locations. Each narrow concentric ring is called a *track*. This discretized environment is *topologically* equivalent to the surface of a discretized cylinder of height k partitioned into k narrow rings of length n which are layered on top of each other. For example, the environment of Figure 4.2 corresponds to $k = 3, n = 8$ (3 tracks of length 8). The coordinate (x, y) refers to the xth location on the yth track (which can also be seen as the xth location of a ring of length n wrapped around the cylinder at height y). We define $\forall x, (x+n, y) \equiv (x, y)$.

A swarm of m identical agents, or "locusts," which we label $A_1, \ldots, A_m$, are dispersed at arbitrary locations move autonomously at discrete time steps $t = 0, 1, \ldots$. A given location (x, y) can contain at most one locust. Each locust A_i is initiated with either a "clockwise" or "counterclockwise" *heading*, which determines their present direction of motion. We define $b(A_i) = 1$ when A_i has clockwise heading, and $b(A_i) = -1$ when A_i has counterclockwise heading.

The locusts move synchronously at discrete time steps $t = 0, 1, \ldots$. At every time step, locusts try to take a step in their direction of motion: if a locust A is at (x, y), it will attempt to move to $(x + b(A), y)$. A clockwise movement corresponds to adding 1 to x, and a counterclockwise movement corresponds to subtracting 1. The locusts have physical dimension, so if the location a locust attempts to move to already contains another locust at the beginning of the time step, the locust instead stays put. If A_i and A_j are both attempting to move to the same location, one of them is chosen uniformly at random to move to the location and the other stays put.

Locusts that are adjacent exert pressure on each other to change their heading: if A_i has a clockwise heading and A_j has a counterclockwise heading, and they lie on the coordinates (x, y) and $(x + 1, y)$ respectively, then at the end of the current time step, one locust (chosen uniformly at random) will flip its heading to the other locust's heading[1]. Such an event is called a **conflict** between A_i and A_j. A conflict is "won"

[1] An equivalent way to model these dynamics is as follows: at the start of a conflict, each of the two locusts uniformly samples a random number $r_i, r_j \in (0, 1)$ called 'pressure'. The locust with lower

by the locust that successfully converts the other locust to their heading.

Let A be a locust at (x, y). If the locust A has clockwise heading, then the *front* of A is the first locust after A in the clockwise direction, and the *back* of A is the first locust in the counterclockwise direction. The reverse is true when A has counterclockwise heading. Formally, let $i > 0$ be the smallest positive integer such that $(x + b(A)i, y)$ contains a locust, and let $j > 0$ be the smallest positive integer such that $(x - b(A)j, y)$ contains a locust. The *front* of A is the locust in $(x + b(A)i, y)$ and the *back* of A is the locust in $(x - b(A)j, y)$. The locusts in the front and back of A are denoted $A^{\rightarrow}$ and $A^{\leftarrow}$ respectively, and are called A's *neighbours*; these are the locusts that are directly in front of and behind A. Note that when a track has two or less locusts, $A^{\rightarrow} = A^{\leftarrow}$. When a track has one locust, $i = j = n$ and so $A = A^{\rightarrow} = A^{\leftarrow}$.

At any given time step, besides moving in the direction of their heading within their track, a locust A at (x, y) can switch tracks, moving vertically from (x, y) to $(x, y + 1)$ or $(x, y - 1)$ (unless this would cause it to go above track k or below track 1). Such vertical movements occur *after* the horizontal movements of locusts along the tracks, but on the same time step where those horizontal movements took place. Locusts are incentivized to move vertically when this enables them to avoid changing their heading ("inertia"). Specifically, A may move to the location $E = (x, y \pm 1)$ at time t when:

1. At the beginning of time t, A and $A^{\rightarrow}$ are not adjacent to each other and $b(A) \neq b(A^{\rightarrow})$.

2. Once A moves to E, the updated $A^{\leftarrow}$ and $A^{\rightarrow}$ in the new track will have heading $b(A)$.

3. No locust will attempt to move horizontally to E at time $t + 1$.

Condition (1) states that there is an imminent conflict between A and $A^{\rightarrow}$ which is bound to occur. Condition (2) guarantees that, by changing tracks to avoid this conflict, A is not immediately advancing towards another collision; A's new neighbours will have the same heading as A. Condition (3) guarantees that the location A wants to move to on the new track is not being contested by another locust already on that track. Together, these conditions mean that locusts only change tracks if this results in avoiding collisions and in "swarming" together with other locusts marching in the same direction of motion. If a locust cannot sense that all three conditions (1), (2) and (3) are fulfilled, it does not switch tracks.

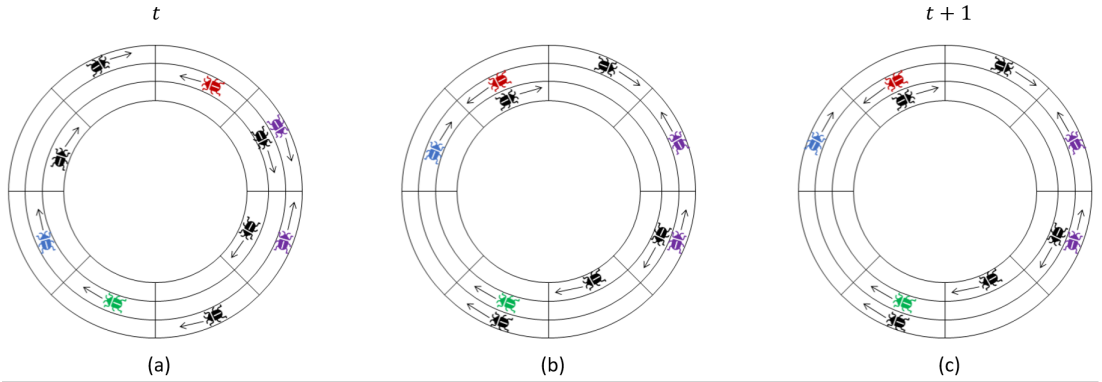
Besides these conditions, we make no assumptions about *when* locusts move vertically. In other words, locusts do not always need to change tracks when they are allowed to by rules (1)-(3); they may do so arbitrarily, say with some probability q or according to any internal scheduler or algorithm, and we may impose visibility range constraints on the locusts such that they only switch tracks when they can *see* that rules

pressure "loses" the conflict and changes its heading (noting that the probability of $r_i = r_j$ is 0).

(1)-(3) are fulfilled. We do not determine in any sense the times when locusts move between tracks–but only determine the preconditions required for such movements; our results in the following sections remain true regardless. This makes our results general in the sense that they hold for many different track-switching "swarming" rules, as long as those rules do not break the conditions (1)-(3).

Figure 4.2 illustrates one time step of the model, split into horizontal and vertical movement phases.

Figure 4.2 One step of the locust model with $k = 3$, $n = 8$ split into horizontal and vertical movements. (a) shows the initial configuration at the beginning of the current time step t, (b) illustrates changes to the configuration after conflicts and horizontal movements, and (c) is the configuration at the beginning of time $t + 1$ (or equivalently the end of time t) after vertical movements. The *front* and *back* of the blue locust are the red and green locusts respectively. The purple locusts conflict with each other. Since conditions (1)-(3) are fulfilled, the blue locust may switch tracks, and it does so in the illustration.



In order to slightly simplify our analysis of the model, we assume that every track has at least 2 locusts at all times, although our results remain true without this assumption.

Although we work in a discrete time model where movement is instantaneous, it is helpful for the sake of formal analysis to define *the beginning* of a time step as the configuration of the swarm at that time step before any locusts moved, and *the end* of a time step as the configuration at that time step after all locust movements are complete. Somewhat idiosyncratically, the end of time t is precisely the beginning of time $t+1$ - both terms refer to the same thing. By default and unless stated otherwise, the words "time step t" refer to the beginning of that time step.

4.4 Stabilization analysis

We will mainly be interested studying the stability of the headings of the locusts over time. Does the model reach a point where the locusts stabilize and stop changing their heading? If so, are their headings all identical? How long does it take?

In the case of a single track ($k = 1$), we shall see that the locusts all eventually

stabilize with identical heading, and bound the expected time for this to happen in terms of m and n. In the multi-track case, we shall see that the locusts stabilize and agree on a heading *locally* (i.e., all locusts *on the same track* eventually have identical heading and thereafter never change their heading), and bound the expected time to stabilization in terms of m, n, k. In the multi-track case, we show further that adding a small probability of "erratic" track-switching behaviour to the model induces *global* consensus: all locusts across all tracks eventually have identical heading.

4.4.1 Locusts on narrow ringlike arenas ($k = 1$)

We start by studying the case $k = 1$, that is, we study a swarm of m locusts marching on a single track of length n. Throughout this section, we assume this is the case, except in Definition 4.4.1, which is also used in later sections.

For the rest of this section, let us call the swarm *non-stable* at time t if there are two locusts A_i and A_j such that $b(A_i) \neq b(A_j)$; otherwise, the swarm is *stable*. A swarm which is stable at time t remains stable thereafter. We wish to bound the number of time steps it takes for the system to become stable, which we denote T_{stable}. Our goal is to prove Theorem 4.1, which tells us that the expected time to stabilization grows quadratically in the number of locusts m, and linearly in the track length n.

Theorem 4.1. *For any configuration of m locusts on a ring with a single track, $\mathbb{E}[T_{stable}] \leq m^2 + 2(n - m)$. This bound is asymptotically tight: there are initial locust configurations for which $\mathbb{E}[T_{stable}] = \Omega(m^2 + n - m)$.*

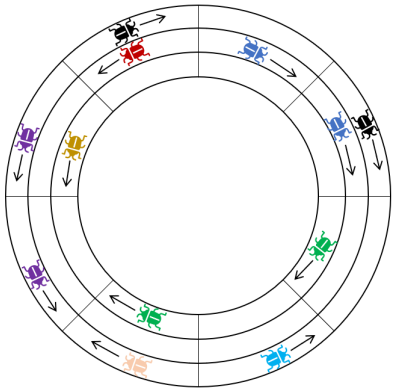
Theorem 4.1 tells us that all locusts must have identical bias within finite expected time. This fact in isolation (without the time bounds in the statement of the theorem) is relatively straightforward to prove, by noting that the evolution of the locusts' headings and locations can be modelled as a finite Markov chain, and the only absorbing classes in this Markov chain are ones in which all locusts have the same heading (see [GS12]).

Next we define *segments*: sets of consecutive locusts on the same track which all have the same heading. This allows us to partition the swarm into segments, such that every locust belongs to a unique segment (see Figure 4.3). Although this section focuses on the case of a single track (and claims in this section are made under the assumption that there is only a single track), the definition is general, and we will use it in subsequent sections.

Definition 4.4.1. Let A be a locust for which $b(A^{\leftarrow}) \neq b(A)$ at time t, and consider the sequence of locusts $B_0 = A$, $B_{i+1} = B_i^{\rightarrow}$. Let B_q be the first locust in this sequence for which $b(B_q) \neq b(B_0)$. The set $\{B_0, B_1, \dots B_{q-1}\}$ is called the **segment** of the locusts $B_0, \dots B_{q-1}$ at time t. The locust B_{q-1} is called the **segment head**, and A is called the **segment tail** of this segment.

Only locusts which are segment heads at the beginning of a time step can change their heading by the end of that time step. When the heads of two segments are adjacent

Figure 4.3 A locust configuration with $n = 8, k = 3$. Locusts are colored based based on the segment they belong to (Definition 4.4.1). There are 8 segments in total.



to each other, the resulting conflict causes one to change its heading, leave its previous segment, and instead become part of the other segment. If the head of a segment is also the tail of a segment, the segment is eliminated when it changes heading. Two segments separated by a segment of opposite heading merge if the opposite-heading segment is eliminated, which decreases the number of segments by 2. No other action by a locust can change the segments. Hence, the number of segments and segment tails can only decrease.

Since our model is stochastic, different sequences of events may occur and result in different segments. However, by the above argument we can conclude that in any such sequence of events, there must always exist at least one locust which remains a segment tail at all times $t < T_{stable}$ and never changes its heading (since at least one segment must exist as long as $t < T_{stable}$). Arbitrarily denote one such segment tail "A_W".

Definition 4.4.2. The segment of A_W at the beginning of time t is called the **winning segment** at time t, and is denoted $SW(t)$. The head of $SW(t)$ is labelled $H_W(t)$. For convenience, if at time t_0 the swarm is stable (i.e. $t_0 \geq T_{stable}$), then we define $SW(t_0)$ as the set that contains all m locusts.

Lemma 4.4.3. *The expected number of time steps $t < T_{stable}$ in which $|SW(t)|$ changes is bounded by m^2.*

Proof Let C_m denote the number of changes to the size of $SW(t)$ that occur before time T_{stable}. Note that T_{stable} is the first time step where $|SW(t)| = m$. $|SW(t)|$ can only decrease, by 1 locust at a time, if $H_W(t)$ conflicts with another locust and loses. $|SW(t)|$ can increase in several ways, for example when it merges with other segments. In particular, $|SW(t)|$ increases by at least 1 whenever $H_W(t)$ conflicts with a locust and wins, which happens with probability at least $\frac{1}{2}$. Hence, whenever $SW(t)$ changes in size, it is more likely to grow than to shrink. We can bound $E[C_m]$ by comparing the growth of $|SW(t)|$ to a random walk with absorbing boundaries at 0 and m:

Consider a random walk on the integers which starts at $|SW(0)|$. At any time step t, the walker takes a step left with probability $\frac{1}{2}$, otherwise it takes a step right. If the walker reaches either 0 or m, the walk ends. Denote by C_m^* the time it takes the walk to end. Using *coupling* (cf. [Lin02]), we see that $\mathbb{E}[C_m] \leq \mathbb{E}[C_m^* | \text{the walker never reaches 0}]$, since per the previous paragraph, $|SW(t)|$ clearly grows at least as fast as the position of the random walker (note that $|SW(t)| > 0$ is always true, which is analogous to the walker never reaching 0).

Let us show how to bound $\mathbb{E}[C_m^* | \text{the walker never reaches 0}]$. Since the walk is memoryless, we can think of this quantity as the number of steps the random walker takes to get to m, assuming it must move right when it is at 0, and assuming the step count restarts whenever it moves from 0 to 1. If we count the steps without resetting the count, we get that this is simply the expected number of steps it takes a random walker walled at 0 to reach position m, which is at most m^2 (cf. [AF95]). Hence $\mathbb{E}[C_m^* | \text{the walker never reaches 0}] \leq m^2$. ■

Lemma 4.4.4. *The expected number of time steps $t < T_{stable}$ in which $|SW(t)|$ does not change is bounded by $2(n - m)$.*

Lemma 4.4.4 will require other lemmas, and some new definitions to prove.

Definition 4.4.5. Let A and B be two locusts or two locations which lie on the same track. The clockwise distance from A to B at time t is the number of clockwise steps required to get from A's location to B's location, and is denoted $dist^c(A, B)$. The counterclockwise distance from A to B is denoted $dist^{cc}(A, B)$ and equals $dist^c(B, A)$.

For the rest of this section, let us assume without loss of generality that the winning segment's tail A_W has clockwise heading. Label the empty locations in the ring at time $t = 0$ (i.e., the locations not containing locusts at time $t = 0$) as $E_1, E_2, \ldots E_{n-m}$, sorted by their counterclockwise distance to A_W at time $t = 0$, such that E_1 minimizes $dist^{cc}(E_i, A_W)$, E_2 has the second smallest distance, and so on. We will treat these empty locations as having persistent identities: whenever a locust A moves from its current location to E_i, we will instead say that A and E_i *swapped*, and so E_i's new location is A's old location.

We say a location E_i is *inside* the segment $SW(t)$ at time t if the two locusts which have the smallest clockwise and counterclockwise distance to E_i respectively are both in $SW(t)$. Otherwise, we say that E_i is *outside* $SW(t)$. A locust or location A is said to be *between* E_i and E_j, $j > i$, if $dist^c(E_i, A) < dist^c(E_i, E_j)$.

Definition 4.4.6. All empty locations are initially **blocked**. A location E_i becomes **unblocked** at time $t + 1$ if all empty locations E_j such that $j < i$ are unblocked at time t, and a locust from $SW(t)$ swapped locations with E_i at time t. Once a location becomes **unblocked**, it remains that way forever.

Lemma 4.4.7. *There is some time step $t^* \leq n - m$ such that:*

1. *Every blocked empty location E is outside $SW(t^*)$ (if any exist)*

2. *At least t^* empty locations are unblocked.*

Proof If E_1 is outside $SW(0)$, then the same must be true for all other empty locations, so $t^* = 0$ and we are done. Otherwise, E_1 becomes unblocked at time $t = 1$. If E_i becomes unblocked at time t, then at time t, it cannot be adjacent to E_{i+1}, since the locust that swapped with E_i in the previous time step is now between E_i and E_{i+1}. By definition, there are no empty locations E_j between E_i and E_{i+1}. Consequently, if E_{i+1} is inside $SW(t)$ at time t, it will swap with a locust of $SW(t)$ at time t, and become unblocked at time $t+1$. If E_{i+1} is outside the segment at time t, it will become unblocked at the first time step $t' > t$ that begins with E_{i+1} inside $SW(t')$. Hence, if E_i becomes unblocked at time t, then E_{i+1} becomes unblocked at time $t+1$ or E_{i+1} is outside $SW(t+1)$ at time $t+1$.

Let t^* be the smallest time where there are no blocked empty locations inside $SW(t^*)$. By the above, at every time step $t \leq t^*$ an empty location becomes unblocked, hence there are at least t^* unblocked empty locations at time t^*. Also, since there are $n - m$ empty locations, this implies $t^* \leq n - m$. ■

Lemma 4.4.8. *There is no time $t < T_{stable}$ where an unblocked location is clockwise-adjacent to $H_W(t)$ (i.e., there is no time t where an unblocked empty location E is located one step clockwise from $H_W(t)$).*

Proof First consider what happens when E_1 becomes unblocked: it swaps its location with a locust in $SW(t)$, and since E_1 is the clockwise-closest empty location to A_W, the entire counterclockwise path from E_1 to A_W consist only of locusts from $SW(t)$. Hence E_1 will move counterclockwise at every time step, until it swaps with A_W. Once it swaps with A_W, E_1 will not swap with another locust at all times $t < T_{stable}$, since for that to occur we must have that $b(A_W^{\leftarrow}) = b(A_W)$, which is impossible since by definition A_W remains a segment tail until $t = T_{stable}$. E_1 does not swap with $H_W(t)$ while E_1 moves counterclockwise towards A_W nor after E_1 and A_W swap as long as the swarm is unstable, hence there is no time step $t < T_{stable}$ when E_1 is unblocked and swaps with $H_W(t)$.

Now consider E_2. E_2 becomes unblocked at least one time step after E_1, and there is at least one locust in $SW(t)$ which is between E_1 and E_2 at the time step E_1 becomes unblocked (in particular, the locust in $SW(t)$ that swapped with E_1 must be between E_1 and E_2 at that time). Since E_1 subsequently moves towards A_W at every time step until they swap, E_2 cannot become adjacent to E_1 until they both swap with A_W. Hence the location one step counterclockwise to E_2 must always be a locust until E_2 swaps with A_W, meaning that similar to E_1, E_2 also moves counterclockwise towards A_W at every time step after E_2 becomes unblocked until they swap locations. Consequently, just like E_1, there is no time step $t < T_{stable}$ when E_2 is unblocked and swaps with $H_W(t)$.

More generally, by a straightforward inductive argument, the exact same thing is true of E_i: once it becomes unblocked, it moves counterclockwise towards A_W at every time step until it swaps with A_W. Thus, upon becoming unblocked, E_i does not swap with $H_W(t)$ as long as $t < T_{stable}$.

Using Lemmas 4.4.7 and 4.4.8, let us prove Lemma 4.4.4.

Proof If, at the beginning of time step t, $H_W(t)$ is adjacent to a locust from a different segment, then $|SW(t)|$ will change at the end of this time step due to the locusts' conflict. Hence, to prove Lemma 4.4.4, it suffices to show that out of all the time steps before time T_{stable}, $H_W(t)$ is not adjacent to the head of a different segment in at most $2(n-m)$ different steps in expectation.

If all empty locations are unblocked at time $n-m$, then by Lemma 4.4.8, $H_W(t)$ conflicts with the head of another segment at all times $t \geq n-m$. Therefore, $|SW(t)|$ will change at every time step $n-m < t < T_{stable}$, which is what we wanted to prove.

If there is a blocked location at time $n-m$, then by Lemma 4.4.4, there must be some time $t^* \leq n-m$ where at least t^* empty locations are unblocked and all blocked empty locations are outside $SW(t^*)$. Let E_j be the minimal-index blocked location which is outside $SW(t^*)$ at time t^*. Since there are no blocked empty locations inside $SW(t^*)$, all locations E_i with $i < j$ are unblocked. Hence, E_j will become unblocked as soon as it swaps with the head of the winning segment. Since (by the clockwise sorting order of $E_1, E_2, \ldots$) E_{j+1} cannot swap with the winning segment head before E_j is unblocked, E_{j+1} will also become unblocked after the first time step where it swaps the winning segment head. The same is true for $E_{j+2}, \ldots E_{n-m}$. Hence, every empty location that $H_W(t)$ swaps with after time t^* becomes unblocked in the subsequent time step. By Lemma 4.4.4, the total swaps $H_W(t)$ could have made before time T_{stable} is thus most $t^* + (n-m-j) \leq n-m$. Whenever an empty location is one step clockwise from $H_W(t)$, they will swap with probability at least 0.5 (the swap is not guaranteed, since it is possible the location is also adjacent to the head of another segment, and hence a tiebreaker will occur in regards to which segment head occupies the empty location in the next time step). Consequently, the expected number of time steps $H_W(t)$ is not adjacent to the head of another segment is bounded by $2(n-m)$.

The proof of Theorem 4.1 now follows.

Proof Lemma 4.4.4 tells us that before time T_{stable}, $|SW(t)|$ does not change in at most $2(n-m)$ time steps in expectation, whereas Lemma 4.4.3 tells us that the expected number of changes to $|SW(t)|$ before time T_{stable} is at most m^2. Hence, for any configuration of m locusts on a ring of track length n, $\mathbb{E}[T_{stable}] \leq m^2 + 2(n-m)$.

Let us now show a locust configuration for which $\mathbb{E}[T_{stable}] = \Omega(m^2+n)$, so as to asymptotically match the upper bound we found. Consider a ring with $k = 1$, m divisible by 2, and an initial locust configuration where locusts are found at coordinates $(0,1), (1,1), \ldots (m/2, 1)$ with clockwise heading and at $(-1,1), (-2,1), \ldots (-m/2-1, 1)$

with counterclockwise heading, and the rest of the ring is empty. This is a ring with exactly two segments, each of size $m/2$. Since after every conflict, the segment sizes are offset by 1 in either direction, the expected number of conflicts between the heads of the segments that is necessary for stabilization is equal to the expected number of steps a random walk with absorbing boundaries at $m/2$ and $-m/2$ takes to end, which is $m^2/4$ (see [Eps12]). Since the heads of the segments start at distance $n - m$ from each other, it takes $\Omega(n - m)$ steps for them to reach each other. Hence the expected time for this ring to stabilize is $\Omega(m^2 + n - m)$.

4.4.2 Locusts on wide ringlike arenas ($k > 1$)

Let us now investigate the case where m locusts are marching on $k > 1$ tracks of length n. The first question we should ask is whether, just as in the case of the $k = 1$ setting, there exists some time T where all locusts have identical heading. The answer is "not necessarily": consider for example the case $k = 2$ where on the $k = 1$ track, all locusts march clockwise, and on the $k = 2$ track, all locusts march counterclockwise. According to the track-switching conditions (Section 4.3), no locust will ever switch tracks in this configuration, hence the locusts will perpetually have opposing headings. As we shall prove in this section, swarms stabilize *locally*–meaning that eventually, all locusts *on the same track* have identical heading, but this heading may be different between tracks.

Let us say that the yth track is stable if all locusts whose location is $(\cdot, y)$ have identical heading. Note that once a track becomes stable, it remains this way forever, as by the model, the only locusts that may move into the track must have the same heading as its locusts. Let T_{stable} be the first time when every all the k tracks are stable. Our goal will be to prove the following asymptotic bounds on T_{stable}:

Theorem 4.2. $\mathbb{E}[T_{stable}] = \mathcal{O}(\min(\log(k)n^2, mn))$.

Recalling Definition 4.4.1, each locust in the system belongs to some segment. Each track has its own segments. Locusts leave and join segments due to conflicts, or when they pass from their current segment to a track on a different segment. In this section, we will treat segments as having persistent identities, similar to SW in the previous section. We introduce the following notation:

Definition 4.4.9. Let S be a segment whose tail is A at some time t_0. We define $S(t)$ to be the segment whose tail is A at the beginning of time t. If A is not a segment tail at time t, then we will say $S(t) = \emptyset$ (this can happen once A changes its heading or moves to another track, or due to another segment merging with $S(t)$ which might cause $b(A^{\leftarrow})$ to equal $b(A)$, thus making A no longer the tail).

Furthermore, define S_1 to be the segment tail of S and $S_{i+1} = S_i^{\rightarrow}$.

Let us give a few examples of the notation in Definition 4.4.9. Suppose at time t_1 we have some segment S. Then the tail of S is S_1, and the head is $S_{|S|}$. $S(t)$ is the

segment whose tail is S_1 at time t, hence $S(t_1) = S$. Finally, $S(t)_{|S(t)|}$ is the head of the segment $S(t)$.

In the $k > 1$ setting, locusts can frequently move between tracks, which complicates our study of T_{stable}. Crucially, however, the number of segments on any individual track is non-increasing. This is because, first, as shown in the previous section, locusts moving and conflicting on the same track can never create new segments. Second, by the locust model, locusts can only move into another track when this places them between two locusts that already belong to some (clockwise or counterclockwise) segment.

That being said, locusts moving in and out of a given track makes the technique we used in the previous section unfeasible. In the following definitions of *compact* and *deadlocked* locust sets, our goal is to identify configurations of locusts on a given track which locusts cannot enter from another track. Such configurations can be studied locally, focusing only on the track they are in. In the next several lemmas, we will bound the amount of time that can pass without either the number of segments decreasing, or all segments entering into deadlock.

Definition 4.4.10. We call a sequence of locusts $X_1, X_2, \ldots$ **compact** if $X_{i+1} = X_i^{\rightarrow}$ and either:

1. every locust in X has clockwise heading and for every $i < |X|$, $dist^c(X_i, X_{i+1}) \leq 2$, or
2. every locust in X has counterclockwise heading and for every $i < |X|$, $dist^{cc}(X_i, X_{i+1}) \leq 2$.

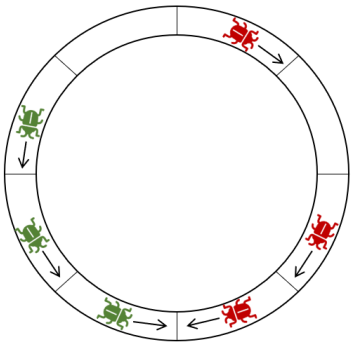
An unordered set of locusts is called compact if there exists an ordering of all its locusts that forms a compact sequence.

Definition 4.4.11. Let $X = \{X_1, X_2, \ldots X_j\}$ and $Y = \{Y_1, Y_2, \ldots Y_k\}$ be two compact sets, such that the locusts of X have clockwise heading and the locusts of Y have counterclockwise heading. X and Y are **in deadlock** if $dist^c(X_j, Y_k) = 1$. (See Figure 4.4)

A compact set of locusts X is essentially a platoon of locusts all on the same track which are heading in one direction, and are all jammed together with at most one empty space between each consecutive pair. As long as X remains compact, no new locusts can enter the track between any two locusts of X, because the model states that locusts do not move vertically into empty locations to which a locust is attempting to move horizontally, and the locusts in a compact set are always attempting to move horizontally to the empty location in front of them.

Definition 4.4.12. A maximal compact set is a set X such that for any locust $A \notin X$, $X \cup A$ is not compact.

Figure 4.4 Two segments in deadlock, colored green and red (Definition 4.4.11).



A straightforward observation is that locusts can only belong to one maximal compact set:

Observation 4.4.13. Let A be a locust. If X and Y are maximal compact sets containing A, then $X = Y$.

Lemma 4.4.14. *Let X and Y be two sets of locusts in deadlock at the beginning of time t. Then at every subsequent time step, the locusts in $X \cup Y$ can be separated into sets X' and Y' that are in deadlock, or the locusts in $X \cup Y$ all have identical heading.*

Proof Let $X = \{X_1, X_2, \ldots X_j\}$ and $Y = \{Y_1, Y_2, \ldots Y_k\}$ be compact sets such that $X_{i+1} = X_i^{\rightarrow}$, $Y_{i+1} = Y_i^{\rightarrow}$. It suffices to show that if X and Y are in deadlock at time t, they will remain that way at time $t+1$, unless $X \cup Y$'s locusts all have identical heading. Let us assume without loss of generality ("w.l.o.g.") that X has clockwise heading, and therefore Y has counterclockwise heading. By the definition of deadlock, at time t, X_j and Y_k conflict, and the locust that loses joins the other set. Suppose w.l.o.g. that X_j is the locust that lost. If $|X| = 1$, then the locusts all have identical heading, and we are done. Otherwise, set $X' = \{X_1, \ldots X_{j-1}\}$ and $Y' = \{Y_1, Y_2, \ldots Y_k, X_j\}$. Note that since X and Y are compact at time t, no locust could have moved vertically into the empty spaces between pairs of locusts in $X \cup Y$. Furthermore the locusts of X and Y all march towards X_j and Y_k respectively, hence the distance between any consecutive pair X_i, X_{i+1} or Y_i, Y_{i+1} could not have increased. Thus X' and Y' are compact.

To show that X' and Y' are deadlocked at time $t+1$, we need just to show that $dist^c(X_{j-1}, X_j)$ is 1 at time $t+1$. Since the distances do not increase, if $dist^c(X_{j-1}, X_j)$ was 1 at time t, we are done. Otherwise $dist^c(X_{j-1}, X_j) = 2$ at time t, and since X_j did not move (it was in a conflict with Y_k), X_{j-1} decreased the distance in the last time step, hence it is now 1. ■

Lemma 4.4.15. *Suppose P and Q are the only segments on track $\mathcal{K}$ at time t_0, and P's locusts have clockwise heading. Let $d = dist^c(P_1, Q_1)$. After at most $3d$ time steps, $P(t_0 + 3d)$ and $Q(t_0 + 3d)$ are in deadlock, or the track is stable.*

Proof The track $\mathcal{K}$ consists of locations of the form (x, y) for some fixed y and $1 \leq x \leq n$. For brevity, in this proof we will denote the location (x, y) simply by its horizontal coordinate, i.e., x, by writing $(x) = (x, y)$.

We may assume w.l.o.g. that $t_0 = 0$, and that P_1 is initially at (0). Note that this means Q_1 is at (d) at time 0. If at any time $t \leq 3d$, the track is stable, then we are done, so we assume for contradiction that this is not the case. This means that P_1 and Q_1 do not change their headings before time 3d. This being the case, we get that $dist^c(P_1, Q_1)$ is non-increasing before time 3d. As the segments $P(t)$ and $Q(t)$ move towards each other at every time step $t \leq 3d$, we can consider only the interval of locations $[0, d]$, i.e., the locations $(0), (1), \ldots (d)$. We then define the distance $dist(\cdot, \cdot)$ between two locusts in this interval whose x-coordinates are x_1 and x_2 as $|x_1 - x_2|$.

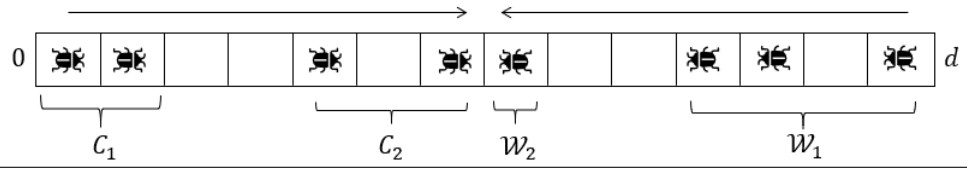
At any time $t \leq 3d$, we may partition the locusts in $[0, d]$ into maximal compact sets of locusts. This partition is unique, by Observation 4.4.13. Let us label the maximal compact sets of locusts that belong to $P(t)$ as $\mathcal{C}_1^t, \mathcal{C}_2^t, \ldots \mathcal{C}_{c_t}^t$, where the segments are indexed from 1 to c_t, sorted by increasing x coordinates, such that $\mathcal{C}_1^t$ contains the locusts closest to (0). Analogously, we label the maximal compact sets that belong to $Q(t)$ as $\mathcal{W}_1^t, \mathcal{W}_2^t, \ldots \mathcal{W}_{w_t}^t$, with indices running from 1 to w_t, sorted by decreasing x-coordinates such that $\mathcal{W}_1^t$ contains the locusts that are closest to (d) (see Figure 4.5). In this proof, the distance between two sets of locusts X, Y, denoted $dist(X, Y)$, is defined simply as the minimal distance between two locusts $A \in X, B \in Y$. Our proof will utilise the functions:

$$\begin{gathered} L_1(t) = \sum_{i=1}^{c_t - 1} dist(\mathcal{C}_i^t, \mathcal{C}_{i+1}^t),\ L_2(t) = \sum_{i=1}^{w_t - 1} dist(\mathcal{W}_i^t, \mathcal{W}_{i+1}^t) \\ L_3(t) = dist(\mathcal{C}_{c_t}^t, \mathcal{W}_{w_t}^t),\ L(t) = L_1(t) + L_2(t) + L_3(t) \end{gathered} \tag{4.1}$$

$L_1(t)$ is the sum of distances between consecutive clockwise-facing sets in the partition at time t. $L_2(t)$ is the sum of distances between the counterclockwise sets. $L_3(t)$ is the distance between the two closest clockwise and counterclockwise facing sets. The function $L(t)$ is the sum of distances between consecutive compact sets in the partition. When $L(t) = 1$, there are necessarily only one clockwise and one counterclockwise facing sets in the partition, which must equal $P(t)$ and $Q(t)$ respectively. Furthermore, $L(t) = 1$ implies that the distance between $P(t)$ and $Q(t)$ is 1. Hence when $L(t) = 1$, $P(t)$ and $Q(t)$ are both in deadlock. The converse is true as well, hence $L(t) = 1$ if and only if $P(t), Q(t)$ are in deadlock. We will use $L(t)$ as a potential or "Lyapunov" function [LL12] and show it must decrease to 1 within 3d time steps. By Lemma 4.4.14, once P and Q are in deadlock they will remain in deadlock until one of them is eliminated, which completes the proof.

Let us denote by $max(X)$ the locust with maximum x-coordinate in X, and by $min(X)$ the locust with minimal x-coordinate. We may also use $max(X)$ and $min(X)$

Figure 4.5 A partition into maximal compact subsets as in our construction. In this configuration, $L_1(t) = 3$, $L_2(t) = 3$, $L_3(t) = 1$, and $L(t) = 7$. Note that although $\mathcal{C}_1, \mathcal{C}_2$ are compact, $P(t) = \mathcal{C}_1 \cup \mathcal{C}_2$ is not compact, and similarly $Q(t)$ is not compact, thus $P(t)$ and $Q(t)$ are not in deadlock and $L(t) \neq 1$.



to denote the x coordinate of said locust. Note that $dist(\mathcal{C}_i^t, \mathcal{C}_{i+1}^t)$ is the distance between $max(\mathcal{C}_i^t)$ and $min(\mathcal{C}_{i+1}^t)$.

Recall that in the locust model, every time step is divided into a phase where locusts move horizontally (on their respective tracks), and a phase where they move vertically. First, let us show that the sum of distances $L_1(t)$ does not increase due to changes in either the horizontal or vertical phase. Since $L_1(t)$ is the sum of distances between compact partition sets whose locusts move clockwise, and for all $\mathcal{C}_i^t$ except perhaps $\mathcal{C}_{c_t}^t$, $max(\mathcal{C}_i^t)$ always moves clockwise, the distance $dist(\mathcal{C}_i^t, \mathcal{C}_{i+1}^t)$ does not increase as a result of locust movements (note that clockwise movements of $max(\mathcal{C}_i^t)$ do not result in a new compact set because the rest of the locusts in C_i^t follow it). Furthermore, since conflicts cannot result in a new maximal compact set in the partition, conflicts do not increase $L_1(t)$. Hence, $L_1(t)$ does not increase in the horizontal phase. In the vertical phase, clockwise-heading locusts entering the track either create a new set in the partition, which does not affect the sum of distances (as they then merely form a "mid-point" between two other maximal compact sets), or they join an existing compact set, which can never increase $L_1(t)$. By the locust model, the only locusts that can move tracks are $max(\mathcal{C}_{c_t}^t)$ and $min(\mathcal{W}_{w_t}^t)$, since these are the only locusts for which the condition $b(A) \neq b(A^{\rightarrow})$ is true, so locusts moving tracks cannot increase $L_1(t)$ either. In conclusion, $L_1(t)$ is non-increasing at any time step. By analogy, $L_2(t)$ is non-increasing.

Similar to L_1 and L_2, the distance $L_3(t)$ cannot increase as a result of locusts entering the track. It can increase as a result of a locust conflict which eliminates either $\mathcal{W}_{w_t}^t$ or $\mathcal{C}_{c_t}^t$, but such an increase is compensated for by a comparable decrease in either $L_1(t)$ or $L_2(t)$. It is also simple to check that, since $P(t)$ and $Q(t)$ are always moving towards each other when they are not in deadlock (i.e., when $L(t) > 1$), there will be at least two compact sets in the partition that decrease their distance to each other, hence L_1, L_2 or L_3 must decrease by at least 1 in the horizontal phase.

To conclude: $L_1(t)$ and $L_2(t)$ are non-increasing. $L_3(t)$ is non-increasing during the horizontal phase and as a result of new locusts entering $\mathcal{K}$. If $L(t) > 1$, $L(t)$ decreases during each horizontal phase. Hence, $L(t)$ decreases in every time step where $L(t) > 1$ and no locusts in $\mathcal{K}$ move to another track.

What happens when locusts in $\mathcal{K}$ do move to another track? As proven, $L_1(t)$

and $L_2(t)$ do not increase. However, the distance $L_3(t)$ will increase, since the only locusts that can move tracks are $max(\mathcal{C}^t_{w_t})$ and $min(\mathcal{W}^t_{c_t})$. It is straightforward to check that when $\mathcal{C}^t_{c_t}$ contains more than one locust, $L_3(t)$ will increase by at most 2 as a result of $max(\mathcal{C}^t_{w_t})$ moving tracks. When $\mathcal{C}^t_{c_t}$ contains exactly one locust, $L_3(t)$ can increase significantly (as $L_3(t)$ then becomes the distance between $C^t_{c_t-1}$ and W_{w_t}), but any increase is matched by the decrease in $L_1(t)$ as a result of $\mathcal{C}^t_{c_t}$ being eliminated. Analogous statement hold for $\mathcal{W}^t_{w_t}$, and hence $L_3(t)$ can increase by at most 2 as a result of one locust moving out of the track. We need to bound, then, the number of locusts in $\mathcal{K}$ that move tracks before time $3d$. We define the potential function $F(t)$:

$$\begin{aligned} F(t) = \sum_{i=1}^{c_t-1}(dist(\mathcal{C}^t_i, \mathcal{C}^t_{i+1}) - 1) + \sum_{i=1}^{w_t-1}(dist(\mathcal{W}^t_i, \mathcal{W}^t_{i+1}) - 1) + |P(t) \cup Q(t)| = \\ = L_1(t) + L_2(t) - c_t - w_t + |P(t) \cup Q(t)| \end{aligned} \tag{4.2}$$

$F(t)$ is the sum of the empty locations between consecutive compact sets in the partition whose locusts have the same heading, plus the number of locusts in $\mathcal{K}$. Note that $F(t) \geq 0$ at all times t. We will show $F(t)$ is non-increasing, and that it decreases whenever a locust leaves the track. Hence, at most $F(0)$ locusts can leave the track.

Let us show that $F(t)$ is non-increasing. We already know L_1 and L_2 are non-increasing. In the horizontal phase, $|P(t) \cup Q(t)|$ is of course unaffected. c_t and w_t can decrease as a result of maximal compact sets merging, hence increasing F, but this can only happen when the distance between two such sets has decreased, hence the resulting increase to F is undone by a decrease in L_1 and L_2. Hence, $F(t)$ does not increase because of locusts' actions during the horizontal phase.

Likewise, locusts leaving $\mathcal{K}$ can decrease c_t or w_t when they cause a maximal compact set to be eliminated, but this is matched by a comparable decrease in L_1 or L_2 which means that F does not increase due to locusts moving out of the track. Furthermore, $|P(t) \cup Q(t)|$ decreases when this happens. Hence, a locust moving out of the track decreases $F(t)$ by at least 1. Finally, let us show that locusts entering the track does not increase $F(t)$.

At time t, locusts can only enter the track at empty locations that are found in intervals of the form $[max(\mathcal{W}^t_i), min(\mathcal{W}^t_{i+1})]$ or $[max(\mathcal{C}^t_{i+1}), min(\mathcal{C}^t_i)]$ for some i. In particular, locusts cannot enter empty locations that are between two locusts belonging to the same compact set (because a locust in that set will always be attempting to move to that location in the next time step, and the model disallows vertical movements to such locations), nor can they enter the track on the empty locations between $min(\mathcal{C}^t_{c_t})$ and $max(\mathcal{W}^t_{w_t})$. Thus, locusts entering the track at time t decrease the amount of empty locations between two clockwise or counterclockwise compact partition sets (and perhaps cause the sets between which they enter to merge into a single compact set). This will always decrease $L_1(t) + L_2(t) - c_t - w_t$ by at least 1 and increase $|P(t) \cup Q(t)|$

by 1. On net, we see that new locusts entering $\mathcal{K}$ either decreases or does not affect F.

In conclusion, $F(t)$ is non-increasing, and any time a locust moves to another track, $F(t)$ decreases by 1. Thus, at most $F(0)$ locusts can move from $\mathcal{K}$ to another track. Recall that locusts moving out of the track can increase $L(t)$ by at most 2. Hence after at most $L(0) + 2F(0) \leq d + 2d = 3d$ time steps, $L(t) = 1$. ■

Lemma 4.4.16. *Let $seg(t)$ denote the set of segments in all tracks at time t. At time $t + 3n$, either every segment is in deadlock with some other segment, or $|seg(t + 3n)| < |seg(t)|$.*

Proof Consider some track $\mathcal{K}$ and a segment P which is in that track at time t. Let us assume that $|seg(t + 3n)| = |seg(t)|$, and show that $P(t + 3n)$ must be in deadlock with another segment. At any time $t' \geq t$, as long as the number of segments on $\mathcal{K}$ does not decrease, the locusts of $P(t')$ will be marching towards locusts of another segment, which we will label $Q(t')$. They cannot collide or conflict with locusts belonging to any segment other than $Q(t')$. Hence, other segments in $\mathcal{K}$ do not affect the evolution of $P(t)$ and $Q(t)$ before time $t + 3n$, and we can assume w.l.o.g. that $P(t)$ and $Q(t)$ are the only segments in $\mathcal{K}$ at time t. Let d be as in the statement of Lemma 4.4.15. Since $n \geq d$, Lemma 4.4.15 tells us that at some time $t \leq t^* \leq t + 3n$, $P(t^*)$ and $Q(t^*)$ must be in deadlock. Since by Lemma 4.4.14, P and Q must remain in deadlock until one of them is eliminated, we see that at time $t + 3n$ they must still be in deadlock, since we assumed $|seg(t)| = |seg(t + 3n)|$.

Theorem 4.3. $\mathbb{E}[T_{stable}] = \mathcal{O}(mn)$

Proof Let $|seg(t)|$ denote the number of segments at time t. $\mathbb{E}[T_{stable}]$ can be computed as the sum of times $\mathbb{E}[T_2 + T_4 + \ldots + T_{|seg(0)|}]$, where T_i is the expected time until the number of segments drops below i, if it is currently i (we increment the index by 2 since segments are necessarily eliminated in pairs).

Let us estimate $E[T_{2i}]$. Suppose that at time t, the number of segments is $2i$. Then after $3n$ steps at most, either the number of segments has decreased, or all segments are in deadlock. There are in total i pairs of segments in deadlock, and as there are m locusts, there must be a pair P, Q that contains at most $\min(m/i, n)$ locusts at time $t + 3n$. By Lemma 4.4.14, P, Q remain in deadlock until either P or Q is eliminated. We can compute how long this takes in expectation, since at every time step after time $t + 3n$, the heads of P and Q conflict, resulting in one of the segments increasing in size and the other decreasing. Hence, the expected time it takes P or Q to be eliminated is precisely the expected time it takes a symmetric random walk starting at 0 to reach either $|P|$ or $-|Q|$, which is $|P| \cdot |Q| \leq \min((\frac{m}{2i})^2, (\frac{n}{2})^2)$. Hence, $E[T_{2i}] \leq 3n + \min((\frac{m}{2i})^2, (\frac{n}{2})^2)$.

Let us first assume $m \geq n$. Using the fact that $\min((\frac{m}{2i})^2, (\frac{n}{2})^2) = (\frac{n}{2})^2$ for $i \leq \lfloor m/n \rfloor$, we have:

$$\begin{aligned}\mathbb{E}[T_2 + T_4 + \ldots + T_{|seg(0)|}] &\leq 3n \cdot \frac{|seg(0)|}{2} + \lfloor m/n \rfloor (\frac{n}{2})^2 + \sum_{i=\lceil m/n \rceil}^{\infty} (\frac{m}{2i})^2 \\ &\leq \frac{3}{2}mn + \frac{1}{4}mn + \frac{1}{4}m^2 \sum_{i=0}^{\infty} (\frac{1}{m/n+i})^2 \\ &\leq \frac{7}{4}mn + \frac{1}{4}m^2(\frac{n^2}{m^2} + \frac{n}{m}) \leq \frac{9}{4}mn\end{aligned} \tag{4.3}$$

Where we used the inequalities $\sum_{i=0}^{\infty}(\frac{1}{m/n+i})^2 \leq \frac{n^2}{m^2} + \int_{i=0}^{\infty}(\frac{1}{m/n+i})^2 = \frac{n^2}{m^2} + \frac{n}{m}$ and $|seg(0)| \leq m$. If $m < n$, by using the identity the identity $\sum_{i=1}^{\infty}(\frac{1}{i})^2 = \frac{\pi^2}{6}$ we get:

$$\mathbb{E}[T_2+T_4+\ldots+T_{|seg(0)|}] \leq 3n \cdot \frac{|seg(0)|}{2} + \sum_{i=1}^{\infty}(\frac{m}{2i})^2 \leq \frac{3}{2}mn + \frac{\pi^2}{24}m^2 \leq (\frac{3}{2} + \frac{\pi^2}{24})mn \tag{4.4}$$

And so we see that $\mathbb{E}[T_{stable}] = \mathcal{O}(mn)$.

Next we wish to show that $E[T_{stable}] = \mathcal{O}(\log(k)n^2)$. For this, we require the following result:

Lemma 4.4.17. *Consider k independent random walks with absorbing barriers at 0 and $2n$, i.e., random walks that end once they reach 0 or $2n$. The expected time until **all** k walks end is $\mathcal{O}(n^2 \log(k))$.*

Proof First, let us set $k = 1$ and estimate the probability that the one walk has not ended by time t. Let P be the transition probability matrix of the random walk, and let $\mathbf{v}$ be the vector describing the initial probability distribution of the location of the random walker. Then $\mathbf{v}P^t$ is the probability distribution of its location after t time steps [LPW09]. The evolution of $\mathbf{v}P^t$ is well-studied and relates to "the discrete heat equation" [Law10]. The probability that the walk has not ended at time t is the sum $\sum_{i=1}^{2n-1}\mathbf{v}(i)$. Asymptotically, this sum is bounded by $\mathcal{O}(\lambda^t)$ where $\lambda = cos(\frac{\pi}{2n})$ is the 2nd largest eigenvalue of P (cf. [Law10]).

Returning to general k, let $\mathcal{T}_k$ be a random variable denoting the time when all k walks end. By looking at the series expansion of $cos(1/x)$, we may verify that for $n > 1$, $cos(\frac{\pi}{2n}) < 1 - \frac{1}{n^2}$. From the previous paragraph, and because the walks are independent, we therefore see that

$$Pr(\mathcal{T}_k \geq t) = 1 - Pr(\mathcal{T}_1 < t)^k = 1 - (1 - \mathcal{O}(\lambda^t))^k = 1 - (1 - \mathcal{O}((1 - \frac{1}{n^2})^t))^k \tag{4.5}$$

Consequently, for $t \gg n^2$, the following asymptotics hold for some constant C:

$$Pr(\mathcal{T}_k \geq t) < 1 - (1 - Ce^{-t/n^2})^k \tag{4.6}$$

Where we used the fact that $(1 + x/n)^n \to e^x$ as $n \to \infty$. Note that $Pr(\mathcal{T}_k \geq t + n^2 \log(C)) < 1 - (1 - e^{-t/n^2})^k$. Hence:

$$\begin{aligned}
\mathbb{E}[\mathcal{T}_k] &= \int_0^\infty Pr(T_k > t)dt \leq n^2 \log(C) + \int_0^\infty 1 - (1 - e^{-t/n^2})^k dt \\
&= n^2 \log(C) + \int_0^\infty 1 - \sum_{j=0}^{k} \binom{k}{j} (-1)^j e^{-tj/n^2} dt \\
&= n^2 \log(C) + - \sum_{j=1}^{k} \binom{k}{j} (-1)^j \int_0^\infty e^{-tj/n^2} dt \\
&= n^2 \log(C) + -n^2 \sum_{j=1}^{k} \binom{k}{j} \frac{(-1)^j}{j} = \mathcal{O}(n^2 \log(k))
\end{aligned} \tag{4.7}$$

Where we used the equality $\sum_{j=1}^{k} \binom{k}{j} \frac{(-1)^j}{j} = -\sum_{j=1}^{k} \frac{1}{j} \approx \log(k)$.

Theorem 4.4. $\mathbb{E}[T_{stable}] = \mathcal{O}(\log(k) \cdot n^2)$

Proof Let $seg_i(t)$ denote the number of segments in track i at time t, and define $\mathcal{M}_t = \max_{1 \leq i \leq k} seg_i(t)$. Let us bound the expected time it takes for $\mathcal{M}_t$ to decrease. Define the set $K(t)$ to be all tracks that have $|\mathcal{M}_t|$ segments at time t. Then $\mathcal{M}_t$ decreases at the first time $t' > t$ when all tracks in $K(t)$ have had their number of segments decrease. We may bound this with the following argument: slightly generalizing Lemma 4.4.16 to hold for subsets of tracks[2] , if $\mathcal{M}_t$ doesn't decrease after $3n$ time steps (i.e., $\mathcal{M}_t = \mathcal{M}_{t+3n}$), all tracks in $K(t + 3n)$ now have all their segments in deadlock. The number of deadlocked segment pairs at every track in $K(t + 3n)$ is $\mathcal{M}_t/2$, so in every such track there is such a pair with at most $2n/\mathcal{M}_t$ locusts. By Lemma 4.4.17, using a similar argument as Theorem 4.3, these pairs of deadlocked segments resolve into a single segment after at most $c \cdot \log(k)(\frac{2n}{M_t})^2$ expected time for some constant c. Hence, the number of expected time steps for $\mathcal{M}_t$ to decrease is bounded above by $3n + c\log(k)(\frac{2n}{M_t})^2$.

T_{stable} is the first time when $\mathcal{M}_t = 0$. Let us assume n is even for simplicity (the computation will hold regardless, up to rounding). We have that $\mathcal{M}_0 \leq n$, and $\mathcal{M}_t$ decreases in leaps of 2 or more (since segments can only be eliminated in pairs). Hence, T_{stable} is bounded by the amount of time it takes $\mathcal{M}_t$ to decrease at most $n/2$ times. By linearity of expectation, this time can be bounded by summing $3n + c\log(k)(\frac{2n}{M_t})^2$ over $M_t = n, n-2, n-4, \ldots 2$:

[2]Lemma 4.4.16 holds not just for the set $seg(t)$ but for the segments in a given subset of tracks, with the proof being virtually identical. Here we apply the Lemma to the subset $K(t + 3n)$.

$$\begin{aligned}\mathbb{E}[T_{stable}] \leq & \frac{n}{2} \cdot 3n + c\log(k)(\frac{2n}{n})^2 + c\log(k)(\frac{2n}{n-2})^2 + \ldots + c\log(k)(\frac{2n}{2})^2 \\ \leq & \frac{3}{2}n^2 + 4c\log(k)n^2 \sum_{i=1}^{\infty}(\frac{1}{2i})^2 = \frac{3}{2}n^2 + \frac{\pi^2}{6}c\log(k)n^2 = \mathcal{O}(\log(k)n^2)\end{aligned} \tag{4.8}$$

As claimed. ■

The proof of Theorem 4.2 follows immediately from Theorems 4.3 and 4.4, by taking the minimum. ■

Erratic track switching and global consensus

Theorem 4.2 shows that, after finite expected time, all locusts on a track have identical heading. This is a stable *local* consensus, in the sense that two different tracks may have locusts marching in opposite directions forever. We might ask what modifications to the model would force a *global* consensus, i.e., make it so that stabilization occurs only when all locusts across *all* tracks have identical heading. There is in fact a simple change that would force this to occur: let us assume that at time step t any locust has some probability of acting "eratically" in either the vertical or horizontal phases:

1. With probability r, a locust might behave erratically in the horizontal phase, staying in place instead of attempting to move according to its heading.

2. With probability p, a locust may behave erratically in the vertical phase, meaning that even if the vertical movement conditions (1)-(3) of the model (see Section 4.3) are not fulfilled, the locust attempts to move vertically to an adjacent empty space on the track above or below them (if such empty space exists).

These behaviours are independent, and so a locust may behave erratically in both the vertical and horizontal phases, in just one of them, or in neither.

The next theorem shows that the existence of erratic behaviour forces a global consensus of locust headings. The goal is to prove that there is some finite time after which all locusts must have the same heading. Note that the bound we find for this time is crude, and is not intended to approximate T_{stable}. We study the question of how p affects T_{stable} empirically in the next section.

Theorem 4.5. *Assuming there is at least one empty space (i.e., $m < nk$), and the probability of erratic track switching is $0 < r, p < 1$, the locusts all have identical heading in finite expected time.*

Proof Our goal is to show that all locusts must have identical heading in finite expected time. We will find a crude upper bound for this time. It suffices to show that as long as there are two locusts with different headings in the system (perhaps not on the same

track), there is a bounded-above-0 probability q that within a some constant, finite number of time steps C (and shall show $C = \mathcal{O}(\log(k)n^2 + nk)$), the number of locusts with clockwise heading will increase. This amounts to showing that there is a sequence of events, each individual event happening with non-zero probability, that culminates in a conflict between two locusts occurring (since any conflict has probability 0.5 of increasing the number of clockwise locusts). Since $q > 0$, the only stable state of locust headings is the state where all locusts have identical heading, as otherwise there is always some probability that all locusts will have clockwise heading after $m \cdot C$ time steps. This completes the proof.

Let us show such a sequence of events. First let us consider the case where there is a track in which two locusts have non-identical headings. In this case, assuming no locusts behave erratically for $\mathcal{O}(\log(k)n^2)$ steps (which occurs with a tiny but bounded-above-0 probability since $p, r > 0$), Theorem 4.2 tells us that in expected $\mathcal{O}(\log(k)n^2)$ steps, locusts on the same track will have identical heading. Hence, there is a sequence of events that happens with non-zero probability which leads to local consensus in the tracks.

If any conflict occurs during this sequence, we are done. Otherwise, we need to show a sequence of events that leads to a conflict, assuming all tracks are stable. The only thing that causes locusts in local consensus to move tracks is erratic behaviour. If two adjacent tracks have locusts with non-identical heading, and there is at least one empty space in one of them, then (since $r > 0$) with some probability within at most n time steps an empty space in one track will be vertically adjacent to a locust in the other track. At this point, with probability p, that locust will move from one track to the other. This creates a situation where in one track there are locusts of different headings again. If the erratic locust moves tracks at the right time, upon moving it will be adjacent to another locust in its new track, whose heading is different. Hence, the erratic locust will enter a conflict in the next time step, which will increase the number of clockwise locusts with probability 0.5.

Now let us consider a pair of two adjacent tracks with locusts of different headings such that there no empty space in one of them. We note that since there is at least one empty location in *some* track, erratic behaviour can cause that empty location to move vertically in an arbitrary fashion until, after at most k movements, it enters a track from the pair. With non-zero probability, this can take at most nk time steps, after which we are reduced to the situation in the previous paragraph.

A pair of adjacent tracks that have locusts with different headings must exist unless there is global consensus. Hence, in every $\mathcal{O}(\log(k)n^2 + nk)$ time steps where there is no global consensus, there is a some probability $q > 0$ that the number of clockwise-heading locusts will increase. ■

Figure 4.6 Simulations of the locust model. The y axis is T_{stable}. Column (a) measures T_{stable} for $k = 1...30$, with n fixed at 30. Column (b) measures T_{stable} for $n = 1...60$, with k fixed at 5. Column (c) measures T_{stable} with $n = 30, k = 5$, and p (the probability of erratic behaviour) going from 0 to 1. The top row measures T_{stable} in sparse locust configurations ($m \approx 0.1n$), while the bottom row does so for dense configurations ($m \approx 0.5n$). The dashed red line estimates T_{stable} when locusts never switch tracks (except while behaving erratically in column c); the blue line estimates T_{stable} when locusts switch tracks as often as the model rules allow. Error bars show the standard deviations.

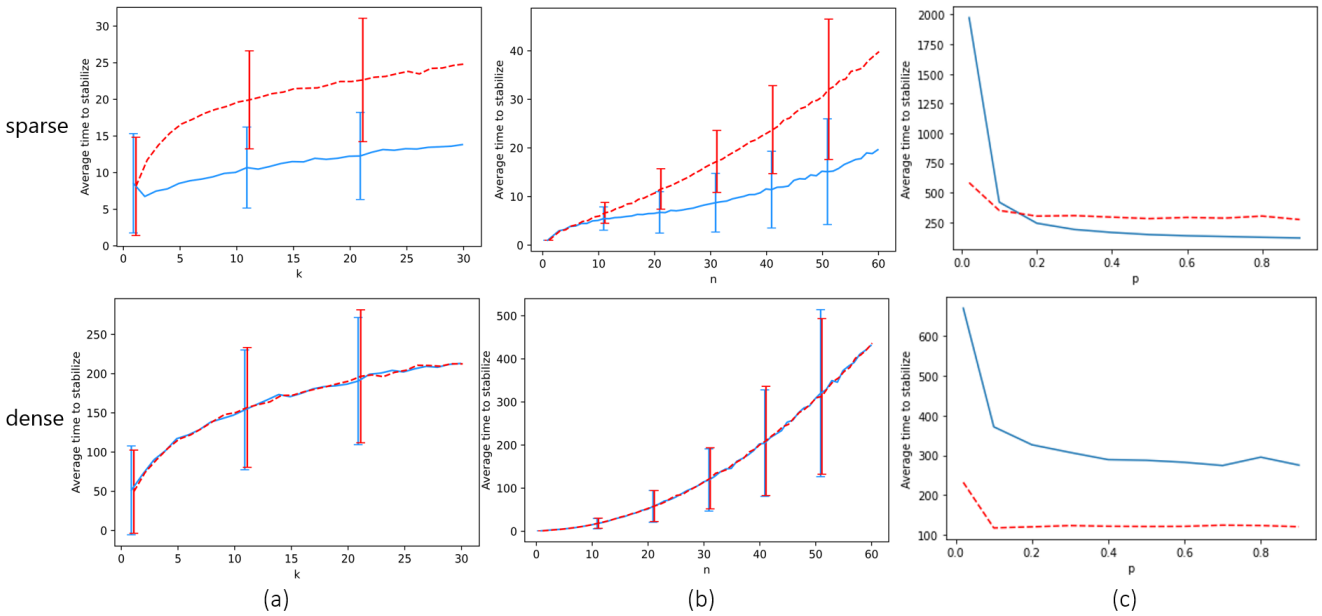
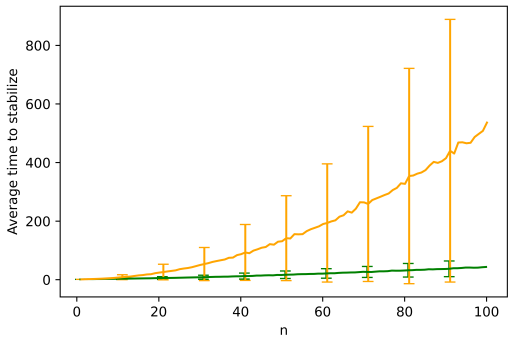


Figure 4.7 Simulations of the locust model fixing $k = 1$ and letting n run from 1 to 100. The y axis is T_{stable}. The orange line denotes dense locust configurations ($m \approx 0.5n$), and the green line denotes sparse configurations ($m \approx 0.1n$). Error bars show the standard deviations.



4.5 Simulation and empirical evaluation

Let us explore some questions about the expected value of T_{stable} through numerical simulations. Certain aspects of the locusts' dynamics were not studied in our formal analysis: the most interesting of which is the helpful effects of track switching on T_{stable}.

Recall that our model allows locusts to switch tracks if this would enable them to avoid a conflict and join a track where *locally*, locusts are marching in their same direction. At least in principle, this seems like it should help our locusts achieve local stability faster, hence decrease T_{stable}. However, recall also that we do not specify *when* locusts switch tracks, which means that some locusts might never switch tracks, or they might choose to do so in the worst possible moments. Hence, the positive effect track-switching usually has on T_{stable} cannot be reflected in the bounds we found for $\mathbb{E}[T_{stable}]$, since these bounds must reflect all possible locust behaviours. Under ordinary circumstances, however, it seems as though frequent track switching should noticeably decrease the time to local stabilization. As we shall see numerically, this is indeed the case. This justifies the track-switching behaviour as a mechanism that, despite being highly local, enables the locusts to achieve local consensus about the direction of motion sooner.

In Figure 4.6, (a) and (b), we measure T_{stable} as it varies with n and k, assuming the probabilities of erratic behaviour are 0 (i.e., $r = p = 0$). We simulate two different locust configurations: a "dense" configuration, and a "sparse" configuration. In the dense configuration, 50% of locations are initiated with a locust, with the locations chosen at random. In the sparse configuration, 10% of locations are initiated with a locust (or slightly more, to guarantee all tracks start with 2 locusts). The locusts are initiated with random heading. We measure the effect of track switching on T_{stable}: the opaque lines measure T_{stable} when locusts switch tracks as often as they can (while still obeying the rules of the model), and the dotted lines measure T_{stable} when locusts never switch tracks. For every value of n, k, we ran the simulation 2000 times and averaged T_{stable} over all simulations.

As we can see, in the sparse configuration, track-switching has a significantly positive effect on time to stabilization. For example, with $k = 30$, $n = 30$, T_{stable} is approximately 13.5 when locusts switch tracks as soon as they can, and approximately 25 when they never switch tracks–nearly double. In the dense configuration, we see that enabling locusts to move tracks has little to no effect, since the locust model rarely allows them to do so due to the tracks being overcrowded.

In column (c) of Figure 4.6, we measure how a non-zero probability p of erratic behaviour affects T_{stable}. We set $r = 0$. As we proved in the previous section, whenever $p > 0$, stabilization requires *global* rather than local consensus. Hence, we cannot directly compare the T_{stable} of these graphs with columns (a) and (b), where T_{stable} measures the time to local consensus. We note that the expectation and variance of T_{stable} approach ∞ as p goes to 0, since when $p = 0$, global stability can never occur in some initial configurations. $\mathbb{E}[T_{stable}]$ decreases sharply as p goes to some critical point around 0.1, and decreases at a slower rate afterwards. It is interesting to note that low probability of erratic behaviour affects $\mathbb{E}[T_{stable}]$ significantly more in the *sparse* configuration, where for $p = 0.02$, if locusts also switch tracks whenever the model allows them, $\mathbb{E}[T_{stable}]$ was measured as being approximately 1974, as opposed to 669 in the dense configuration. One of the core reasons for this seems to be that, in the

sparse configuration, when a locust erratically moves to a track with a lot of locusts not sharing its heading, it will often be able to *non-erratically* move back to its former track, thus preventing locust interactions between tracks of different headings. When we disabled the locusts' ability to switch tracks non-erratically, T_{stable} was significantly smaller in the sparse configuration ($\mathbb{E}[T_{stable}] \approx 232$ for $p = 0.02$).

Based on the above, we make the curious observation that, while non-erratic track switching accelerates local consensus, for some track-switching behaviours, it will in fact decelerate the attainment of global consensus. This is seen by the fact that frequent non-erratic track-switching was helpful in Columns (a) and (b) of Figure 4.6, but increased time to stabilization in Column (c). This is perhaps a very natural observation, because agents that aggressively switch tracks will attempt to avoid conflict as often as possible, whereas conflict is necessary to create global consensus.

To finish this section, we also verify the bounds of Theorem 4.1 by numerical simulation, by fixing $k = 1$ and measuring T_{stable} as n goes from 1 to 100–see Figure 4.7. We again measure both sparse and dense configurations (i.e., $m \approx 0.1n$ and $m \approx 0.5n$ respectively). The average expected time appears asymptotically bounded by m^2, as expected. We also simulated the asymptotic-worst case locust configuration in the proof of Theorem 4.1 (not illustrated in Figure 4.7) and confirmed its stabilization time is asymptotically $\Omega(m^2 + n - m)$, verifying that the bounds of Theorem 4.1 are asymptotically tight.

4.6 Discussion

We studied collective motion in a model of discrete locust-inspired swarms, and bounded the expected time to stabilization in terms of the number of agents m, the number of tracks k, and the length of the tracks n. We showed that when the swarm stabilizes, there must be a local consensus about the direction of motion. We also showed that, when the model is extended to allow a small probability of erratic behaviour to perturb the system, global consensus eventually occurs.

A direct continuation of our work would be to find upper bounds on time to stabilization when there is some probability of erratic behaviour. Furthermore, our empirical simulations suggest several curious phenomena related to erratic behaviour: first, there seems to be a clash between "erratic" and non-erratic, "rational" track-switching, as when locusts switch tracks non-erratically in order to avoid collisions, this seems to accelerate the attainment of local consensus, but mostly hinder the attainment of global consensus. Second, increasing the probability of erratic track-switching p behaviour was helpful in accelerating global consensus up to a point, but in simulations, its impact seemed to fall off past a small critical value of p. In future work, it would be interesting to investigate these "phase transition" aspects of the model.

As discussed in the Related Work section, in [BSC+06; YEE+09] it is observed that at intermediate densities, swarms of locusts exhibit periodic directional switching,

and at low densities the directions of motion are random. Although this phenomenon does not occur in our model, if we assume each locust has a small probability $r > 0$ of randomly flipping their heading at the beginning of a time step, such directional switching becomes possible, with probability inversely proportional to the density (or so we expect). This extension of our model, of course, does not have stable states, thus cannot be studied by the same methods we used in this work. But we would be interested in studying it in terms of the *expected time* the swarm spends in consensus or near-consensus about the direction of motion before directional switching occurs.

For the sake of mathematical theory, we would be very interested in rigorous results established over a fully asynchronous version of this model where locust wake-up times are determined independently. In such a model, the winner of a conflict between two locusts can be determined as the locust that wakes up first (thus exerts pressure on the other locust first), which is perhaps more elegant. We speculate that most of the conclusions will not be majorly affected by transitioning to an asynchronous model.

Although our agent marching model is inspired by experiments on locusts, it can be understood in more abstract terms as a model that describes a situation where many agents that wish to maintain a direction of motion are confined to a small space where they exert pressure on each other. It is natural to ask what kinds of collective dynamics, if any, we should expect when this small space has a different topology; rather than a ringlike arena, we might consider, e.g., a square arena. We believe that rich models of swarm dynamics can be discovered through observing natural organisms exert pressure on each other in such environments. In the introduction, we mentioned points of similarity between our model and models of opinion dynamics. We suspect that these points of similarity will remain in settings with non-ringlike arenas, and might provide a starting point for formally modelling and analysing them.

Chapter 5

Swarm Robotics I: Minimizing Energy in the Multi-Robot Uniform Dispersion Problem

The previous chapter concludes our study of natural algorithms. The second overarching topic of this dissertation is swarm robotics. The objective of swarm robotics is to enable a large group of simple and autonomous mobile robots to work cooperatively towards complex goals. The *ant-robotics paradigm* introduced in Chapter 1 relates swarm robotics to the natural world by connecting robots' capabilities to capabilities exhibited by swarms of social insects. Whereas in the last few chapters our objective was to study the behaviour of (idealized) ants and locusts in the language of this paradigm, we now turn to the problem of *designing* algorithms in the ant-robotics paradigm that are to be used by robotic swarms made by and for human beings.

In the next two chapters we shall investigate a fundamental problem in swarm robotics called *swarm uniform dispersal*, wherein mobile robots are tasked with uniformly covering an a priori unknown discrete environment (e.g. a maze or tunnel). In this chapter, which is based on the paper [AB19b], we are interested in solving uniform dispersal while minimizing the movement and active time of each individual robot, so as to minimize their energy requirements. Our contribution is a local robotic strategy for simply connected grid environments that, by exploiting their topology, achieves optimal makespan (the amount of time it takes to cover the environment) and minimizes the maximal number of steps taken by the individual robots before their deactivation. The robots succeed in discovering optimal paths to their eventual destinations, and finish the covering process in $2V - 1$ time steps, where V is the number of cells in the environment. We further prove an impossibility result which says that a similar algorithm cannot exist in non-simply connected grid environments assuming robots' sensing range is limited. This impossibility result holds even assuming the swarm is composed of non-ant-like robots with infinite memory and communication range.

5.1 Introduction

In many real life scenarios, e.g. mapping or hazard detection, one is interested in deploying agents over an unknown area and covering it for the purposes of sensing or reacting [HMS02]. The use of swarm robotics to solve such problems has many inherent advantages, such as scalability, greater coverage, and autonomy in mission execution. In the *uniform dispersal problem* introduced by [HAB+04], a large number of mobile robots emerge over time from a source or several source locations (called "doors" in the literature), and are tasked to completely cover an unknown environment R by occupying every location and to terminate their work in finite time [BDS08]. The robots must not collide (i.e. two robots must never occupy the same location), nor step outside the boundaries of the environment.

It is often the case, e.g. when the robots are traveling large distances or are airborne, that a lot of energy is required for the sustained activity of robots in the swarm. In this chapter we are interested in solving what is called the uniform dispersal problem in simply connected grid environments while minimizing the movement and active time of each individual robot, so as to minimize the swarm's energy requirements.

Hsiang et al. [HAB+04] [HAB+03] introduced the problem of uniform dispersal in discrete planar domains by mobile robots endowed only with finite memory, local sensors, and local communication. Their DFS-esque "follow the leader" strategy enables robots to cover the environment in optimal time, assuming a synchronous time scheme. Much follow-up work has focused on achieving dispersal with weaker models of robots, e.g. disallowing communication, reducing memory, or assuming asynchronous time [BDS08] [HL17a] [BDS13]. Barrameda et al. [BDS08] have shown that the dispersal problem is intractable under the usual assumptions if the robots are assumed to be oblivious (that is, to possess no persistent states), though there have been attempts to get around this limitation using randomization [HL17b]. It is standard to assume that the robots are moving in a connected grid environment, as any 2D space can be approximated well by pixelation into tiny grid cells of uniform size.

From a theoretical perspective, the problem of dispersing and coordinating mobile robotic agents while minimizing movement or energy has been studied extensively both as a centralized motion planning problem and in distributed sensor networks [DHM09] [LWZ+15] [FS11], and various computational hardness results have been proven in the case of general graph environments [DHM+09]. More broadly, multi-agent scheduling problems have been studied in the presence of energy constraints [HAK18]. Specifically in the context of uniform dispersal for robotic sensors, the question of minimizing travel for orthogonal areas that we here concern ourselves with was discussed in the original paper by Hsiang et. al [HAB+04] and soon after in Stainzberg's doctoral dissertation [Szt03], and more recently in [HB16] and [HL17b].

In recent decades there has been considerable effort dedicated to the algorithmic problems of agent coverage or exploration, wherein a robot or team of robots must com-

pletely explore, occupy, or map an area. Attention has been given to the case of a single robot tasked with visiting every vertex of a graph or grid environment [BS07] [YAK15], to single- and multi-robot path planning [AHKG+08], to natural or pheromone-based computation models [PDE+01] [WLB97] [WLB00], to related formation or dispersal problems [MG07] [CD08], and to a multitude of other topics. We refer the reader to [GC13] or [APB18] for recent surveys. The problem of uniform dispersal distinguishes itself from many of these by its distinctly online nature. The robots emerge onto the environment at different times and must successfully embed themselves into the ongoing exploration effort, without colliding with other robots, and without interrupting the constant outflow of new robots. They must do this under stringent computational, sensory, and communication restrictions–in most recent models, the robots, modelled as finite automata, are not allowed to talk to each other, and cannot even tell the difference between environmental obstacles and the presence of robots active in the formation. We find it fairly surprising that under these restrictions, robots are capable of exploring an entirely unknown environment in theoretically optimal time, as well as (we shall see) walk only in shortest paths to their destinations while doing so.

Much attention has been given to the problem of deployment and coverage in GPS-denied environments, as this may enable the deployment of robotic fleets outside laboratory conditions and their utilization in real world scenarios. Dispersal strategies that operate under stringent restrictions on communication and sensing may be especially relevant to future investigations in this domain. Implementation, however, forms a technical barrier, as when looking at the problem of generating robust uniform coverage from a systems perspective the issues of relative visual localization - range, angular coverage and persistence - become important. There has been progress towards overcoming these barriers in a number of different settings. In [BV12] the authors discuss a visual relative localization method suited for autonomous navigation and obstacle avoidance in indoor environments, for mobile robots with limited computational power. In [SBT+17] the authors present a visual localization method based on an image processing algorithm suitable for use on small quadcopters. The algorithm assumes all the quadcopters have identical but specific markings that ease the localization. These are but examples of the sensors an agent might use when implementing strategies that operate under such restrictions.

Our contribution: Working in a synchronous time setting, Hsiang et al. [HAB+04] pose the problem of minimizing the total and individual number of steps the robots take (the "total travel" and "individual travel"), while achieving optimal makespan–the time before complete coverage of the environment. They describe several algorithms for *general* grid environments that consecutively improve on each other in this respect, but these algorithms do not achieve a global optimum.

We describe a local uniform dispersal strategy that, for *simply connected* grid environments, achieves optimal makespan and minimizes the total travel and maximal individual travel. The strategy's goal is to enable a robot to settle in place as soon as

possible, thereby minimizing the energy consumption. It exploits the ability to decompose simply connected environments into a tree of simply connected sub-environments via "halls"–defined as corners of the environment that also have an obstacle located diagonally opposite to them. We work in a setting similar to [HAB+04], where time is synchronous and robots have local sensors and finite memory. Specifically, the robots require 5 bits of persistent memory (2^5 persistent states), and a visibility span of Manhattan distance 2. As is sometimes assumed, e.g. in [HL17a], they are initialized with a common notion of up, down, left and right. Unlike [HAB+04], our algorithm works without assuming any inter-robot communication capabilities: the robots are only capable of seeing environmental obstacles (including other robots that block them), and are unable to distinguish between kinds of obstacles.

By attempting to restrict their movement to as few directions as possible, our strategy enables the robots to travel in shortest paths from their arrival point to their eventual, a-priori unknown, settling point. The robots finish dispersing in $2V - 1$ time steps, where V is the number of cells in the environment.

On the converse, we show that no local strategy can minimize total travel in general grid environments, even assuming the swarm is composed of non-ant-like robots with infinite memory and communication range.

5.2 Model

Consider the integer grid $\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$, whose vertices are points (x, y) where x and y are both integers, and (x_1, y_1) is connected to (x_2, y_2) if and only if the Manhattan distance $|x_1 - x_2| + |y_1 - y_2|$ is exactly 1. A grid environment or region R is defined as a connected sub-graph of $\mathbb{Z}^2$. The *complement* of R, denoted R^c, is defined as the sub-graph $\mathbb{Z}^2 - R$ of $\mathbb{Z}^2$. We call the vertices of R^c *walls*.

Definition 5.2.1. A region R is said to be **simply connected** if and only if any path $v_1 v_2 \dots v_1$ of vertices in R that forms a closed curve does not surround any vertices of R^c.

In particular, a region R is simply connected if R^c is connected.

A robot is a mobile point in R with limited vision and small finite memory. No two robots may occupy the same location. The visibility range of all robots is assumed to be 2, meaning at every time step, a robot is aware of unoccupied vertices in R that are at a Manhattan distance of 2 or less from it. It infers from this the positions of local *obstacles* (walls or other robots), but cannot distinguish between types of obstacles. All robots have a shared notion of up, down, left and right upon emergence from s.

Time is discretized to steps of $t = 1, 2, \dots$. At every time step, all robots perform a Look-Compute-Move operation sequence, in which they examine their environment and move to a new location based on a computation they perform (a robot may also choose to stay in place - this counts as a move). This occurs synchronously, meaning that all

robots move to their computed next location at the same time. The "beginning" of a time step refers to the configuration of the robots at that time step before the robots move. The "end" of a time step is the configuration at that time step after the robots move.

We denote by $prev(A)$ the position of a robot A at the beginning of the previous time step, and by $next(A)$ its position at the beginning of the next time step.

A given robot is either *active* or *settled*. All robots are initially active, and eventually become settled at the end of some time step. Settled robots never move from their current position.

A unique vertex s in R is designated as the source or "door" vertex. If at the beginning of a time step there is no mobile robot at s, a new robot emerges at s at the end of that time step.

Energy and total travel. The "travel" T_i of the ith robot is the number of time steps t that begin and end with the robot still active. This definition includes steps where the robot does not change location, since we wish to relate travel to energy expenditure (e.g., a quadcopter floating or circling in place is still traveling, and consumes just as much energy). The total travel of the robots is then the sum $\sum T_i$ over all robots, and can be seen as the total amount of energy the robots consume before they settle.

5.3 Find-Corner Depth-First Search

We describe a local rule, "Find-Corner Depth-First Search" (Algorithm 5.1), that enables the robots to disperse over a simply-connected region R. As in [HAB+04], the algorithm has a makespan of $2V - 1$ (where V is the number of cells in R, or equivalently, the total area of R when setting every cell to be a unit square). We note that since at best, robots arrive at s once per two time steps, this is the lowest possible makespan.

The purpose of FCDFS is to minimize the individual travel and total travel of the robots. It does this by ensuring that the path of a robot from s to its eventual destination (the vertex at which it settles) is a shortest path in R.

The idea of the algorithm lies in the distinction between a *corner* and a *hall* (see Figure 5.1 and Figure 5.2):

Definition 5.3.1. A vertex v of a grid environment R is called a **corner** if either:

(a) v has one or zero neighbours in R, or

(b) v has precisely two neighbours u and u' in R, and u and u' have a common neighbour w that is distinct from v.

Definition 5.3.2. A vertex v of R is called a **hall** if it has precisely two neighbours u and u', and u and u' are both adjacent to the same vertex w in R^c.

Figure 5.1 Corners. (Blue vertices are *walls*; vertices in R^c).

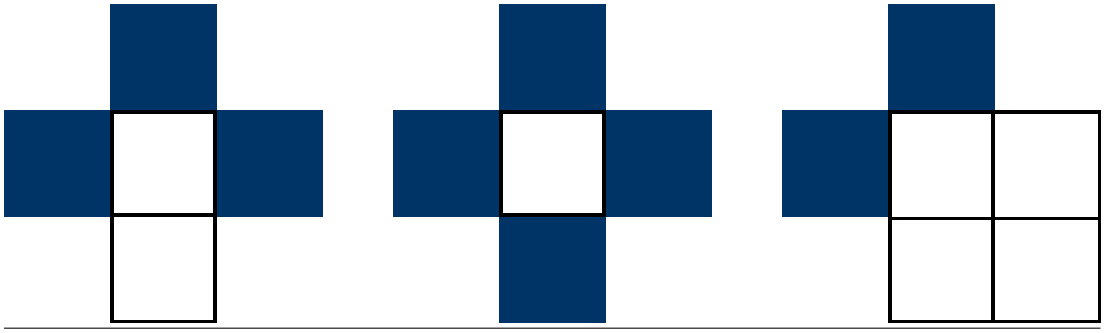
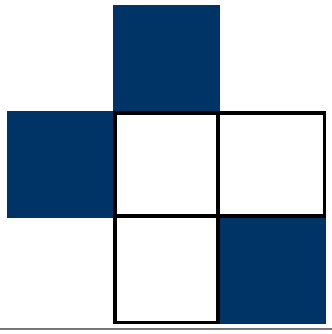


Figure 5.2 A hall.



Essentially, halls are vertices in R that are blocked by walls on two sides, and have an additional wall w diagonal to them. Corners are either dead-ends, or vertices in R that are blocked by walls on two sides, and have a vertex w of R diagonal to them. If v is either a hall or a corner, w is called the "diagonal" of v, and is denoted $diag(v)$. We observe that diagonals are uniquely specified.

Robots executing FCDFS attempt to move only in 'primary' and 'secondary' directions, where the secondary direction is always a 90-degree clockwise rotation of the primary direction (for example "up and right", "right and down", or "down and left"). They may only change their primary direction once they arrive at a hall, and they become settled once both their primary and secondary directions are blocked and they are at a corner.

For the rest of this section, let $R(t)$ be the environment R at time t, i.e. the initial environment R where we have removed from R every vertex that is occupied by a *settled* robot at the beginning of time step t.

A robot at time t is searching for the corners and halls of $R(t)$. However, robots executing FCDFS are unable to distinguish between active robots, and walls or settled robots. Hence, it is important to design the algorithm so that a robot never misidentifies a corner of $R(t)$ as a hall, or vice-versa, due to an active robot (rather than a wall or a settled robot) occupying the diagonal and being identified as an obstacle. For this purpose we enable our robots to remember their two previous locations. We will show that an active robot can occupy the diagonal of a corner at time t if and only if its predecessor occupied this diagonal at time $t-2$, thereby allowing the predecessor to

distinguish between 'real' and 'fake' halls.

Algorithm 5.1 Find-Corner Depth-First Search

Let v be the current location of A.
if every neighbouring vertex of v is occupied **then**
    Settle.
else if A has never moved **then** ▷ Initialization
    Search clockwise, starting from the "up" direction, for an unoccupied vertex, and set primary direction to point to that vertex.
end if
if A can move in its primary direction **then**
    Step in the primary direction.
else if A can step in secondary direction **then**
    Step in the secondary direction.
else ▷ We are at a corner or a hall.
    if $prev(prev(A)) = diag(v) \vee diag(v)$ is unoccupied **then**
        Settle.
    else ▷ We think we are at a hall.
        Set primary direction to point to the neighbour of v different from $prev(A)$.
        Move in the primary direction.
    end if
end if

5.3.1 Analysis

In this section we give an analysis of the FCDFS algorithm. To start, we require some lemmas about corners and halls.

Lemma 5.3.3. *Let c be a corner of a simply connected region R. Then:*

(a) $R - c$ is simply connected.

(b) For any two vertices u, v in $R - c$, the distance between u and v is the same as in R.

Proof Removing c does not affect connectedness, nor does it affect the distance from u to v, as any path going through c can instead go through $diag(c)$. Further, as c is adjacent to two walls, no path in $R - c$ can surround it, so $R - c$ also remains simply connected. ■

An *articulation point* (also known as a separation or cut vertex) is a vertex of a graph whose deletion increases the number of connected components of the graph (i.e. disconnects the graph) [Die17].

Lemma 5.3.4. *The halls of a simply connected region are articulation points.*

Proof Let h be a hall of a simply connected region R. Suppose for contradiction that h is not an articulation point, and let u and u' be the neighbours of h. Then there is a path from u to u' that does not pass through h. Let P be this path, and let P' be the path from u to u' that goes through h.

When embedded in the plane in the usual way, R is in particular a simply connected topological space. The hall h is embedded onto a unit square, whose four corners each touch a wall: three touch the two walls adjacent to h, and the fourth touches $diag(h)$. Joined together to form a closed curve, the paths P and P' form a rectilinear polygon that must contain at least one corner of h in its interior. Hence, the curve PP' contains a part of R^c–and we get a contradiction to the simply connected assumption. (See Figure 5.3).

Figure 5.3 The two possibilities for PP'.

Lemma 5.3.4 indicates that R can be decomposed into a tree structure $T(R)$ as follows: first, delete all halls of R to form separated connected components. Let $C_1, C_2, \ldots, C_n$ be these components, where C_i also includes its adjacent halls. Letting the vertices of $T(R)$ be these components, connect C_i and C_j by an edge if they share a hall. We set C_1 to be the root of the tree, and the connected component containing the door vertex s.

By Lemma 5.3.3, assuming our robots correctly stop only at corners, $R(t)$ can in the same manner be decomposed into a tree $T(R(t))$ whose connected components are $C_1(t), C_2(t), \ldots$. These components are each a sub-graph of a connected component of $T(R)$.

Let $A_1, A_2, \ldots$ denote the robots that emerge from s in the order of arrival. In the next several propositions, we make the *no fake halls at time t* assumption: this is the assumption that for any $t' < t$, at the end of time step t': robots can only become settled at corners of $R(t')$, and can only change primary directions at halls of $R(t')$. We do not include the initialization of a primary direction when a robot arrives at s. We will later show that the "no fake halls" assumption is always true, so the propositions below hold unconditionally.

Proposition 5.3.5. *Assuming no fake halls at time t, a robot A_i active at the beginning of time step t has traveled an optimal path in R from s to its current position.*

Proof By the assumption, the only robots that became settled did so at corners. Consequently, by Lemma 5.3.3, $R(t)$ is a connected graph, and there is a path in $R(t)$ from s to A_i. The path A_i took might not be in $R(t)$, but whatever articulation points (and in particular halls) A_i passed through must still exist, by definition.

Since A_i is active at the beginning of time t, by the algorithm, it has taken a step every unit of time up to t. Until A_i enters its first hall, and between any two halls A_i passes through, it only moves in its primary and secondary directions. This implies that the path A_i takes between the halls of $R(t)$ must be optimal (since it is optimal when embedded onto the integer grid $\mathbb{Z}^2$). We note also that A_i never returns to a hall h it entered a connected component of $R(t)$ from, since the (possibly updated) primary direction pulls it away from h.

We conclude that A_i's path consists of taking locally optimal paths to traverse the connected components of the tree $T(R(t))$ in order of increasing depth. Since in a tree there is only one path between the root and any vertex, this implies that A_i's path to its current location is at least as good as the optimal path in $R(t)$. By Lemma 5.3.3, b, this implies that A_i's path is optimal in R. ■

Corollary 5.1. *Assuming no fake halls at time t,*

(a) For all $i < j$, the distance between the robots A_i and A_j, if they are both active at the beginning of t, is at least $2(j - i)$

(b) No collisions (two robots occupying the same vertex) have occurred.

Proof For proof of (a), note that at least two units of time pass between every arrival of a new robot (since in the first time step after its arrival, a newly-arrived robot blocks s). Hence, when A_j arrives, A_i will have walked an optimal path towards its eventual location at time t, and it will be at a distance of $2(j - i)$ from s. This distance is never shortened up to time t, as A_i will keep taking a shortest path.

(b) follows immediately from (a). ■

From Corollary 5.1 and determinism, we get:

Lemma 5.3.6. *Suppose A_i is active at the beginning of time step t. Assuming no fake halls at time t, $next(A_{i+1}) = prev(A_i)$.*

We note that Lemma 5.3.6 also indicates that if at the beginning of time step t, A_i is active, then A_{i+1} will be active at the beginning of time step $t + 1$.

We can now show that the "no fake halls" assumption is true, and consequently, the propositions above hold unconditionally.

Proposition 5.3.7. *For any t, at the end of time step t: robots only become settled at corners of $R(t)$, and only change primary directions halls of $R(t)$ (not including the primary direction decided at initialization).*

Proof The proof of the proposition is by induction. The base case for $t = 1$ is trivially true.

Suppose that up to time $t-1$, the proposition holds. Note that this means the "no fake halls" assumption holds up to time t, so we can apply the lemmas and propositions above to the algorithm's configuration at the beginning of time t.

We will show that the proposition statement also holds at time t. Let A_i be an active robot whose location at the beginning of t is v. First, consider the case where $v = s$. The algorithm only enables A_i to settle at s if it is surrounded by obstacles at all directions. Any obstacle adjacent to A_i must be a wall of $R(t)$ (as any active robot must be at a distance at least 2 from A_i, due to Corollary 5.1). Hence, if A_i settles at s, s is necessarily a corner, as claimed.

We now assume that $v \neq s$. We separate the proof into two cases:

Case 1: Suppose A_i becomes settled at the end of time step t. Then by the algorithm, at the beginning of t, A_i detects obstacles in its primary and secondary directions. These must be walls of $R(t)$ due to Corollary 5.1, so v is either a corner or a hall of $R(t)$. Since A_i settled, we further know that either $diag(v)$ is empty, or $prev(prev(A_i)) = diag(v)$. In the former case, v is a corner of $R(t)$. In the latter case, we know from Lemma 5.3.6 and from the fact that no collisions occur that the only obstacle detected at $diag(v)$ is A_{i+1}, which is an active robot, so v is again a corner of $R(t)$. In either case a corner is detected and the agent is settled.

Case 2: Suppose A_i changed directions at the end of time step t. Then it sees two adjacent obstacles, and an obstacle at $diag(v)$. As in case 1, we infer that v is either a corner or a hall. If it is a corner, then $diag(v)$ is an active agent. By Corollary 5.1, it is either A_{i+1} or A_{i-1}. It cannot be A_{i+1}, as then A_i's position two time steps ago would have been $diag(v)$, so it would become settled instead of changing directions. It cannot be A_{i-1}, as $diag(v)$ is closer to s than v, and A_{i-1} has arrived earlier than A_i, and has been taking a shortest path to its destination. Hence, $diag(v)$ cannot be an active agent, and v must be a hall as claimed. ■

We have shown that the no fake-hall assumption is justified at all times t, hence we can assume that the propositions introduced in this section hold unconditionally.

Proposition 5.3.8. *Let V be the number of vertices of R. At the end of time-step $2V - 1$, every cell is occupied by a robot.*

Proof Propositions 5.3.5 and 5.3.7 imply that robots take a shortest path in R to their destination. That means that as long as the destination of a robot is not s itself, robots will step away from s one unit of time after they arrive. Until then, this means that robots arrive at s at rate one per two time steps.

Every robot's end-destination is a corner, and by the initialization phase of the algorithm, the destination is never s unless s is completely surrounded. Since there are no collisions, there can be at most V robots in R at any given time. By Lemma 5.3.3, robots that stop at corners keep R connected. Furthermore, every $R(t)$ is a rectilinear polygon, so unless it has exactly one vertex, it necessarily has at least two corners. This means that the destination of every robot is different from s unless s is the only unoccupied vertex. Hence, a robot whose destination is s will only arrive when s is the only unoccupied vertex, and this will happen when V robots have arrived, so after at most $2V-1$ time steps. This is exact, since it is impossible to do better than $2V-1$.■

Propositions 5.3.8 and 5.3.5, alongside the "no fake halls" proof, complete our analysis. They show that FCDFS has a makespan of $2V-1$, and also that the durations of activity of the individual robots are optimal, since every robot travels a shortest path to its destination without stopping.

As every vertex must be occupied for the dispersal to end, a trivial lower bound on the total travel for any dispersal algorithm is $\sum_{v\in R} dist(s,v)$. Since this is achieved by our algorithm, total travel is also minimized.

In practice, the energy savings of our algorithm are dependent on the shape of the environment R. We take as a point of comparison the Depth-First Leader-Follower algorithm of Hsiang et al. [HAB+04]. On a 1-dimensional line of length n, both FCDFS and DFLF require the same total travel, $O(n^2)$, so no improvement is attained. In contrast, on an n-by-n square grid, DFLF requires total travel $O(n^4)$ in the worst case, and FCDFS requires $O(n^3)$ - significantly less. This is because the DFLF strategy starting from a corner might cause the leader, A_1, to "spiral" inwards into the grid, covering every one of its n^2 vertices in n^2-1 moves; the subsequent robot A_i will make n^2-i moves, for a sum total of $O(n^4)$. FCDFS, on the other hand, distributes the path lengths more uniformly. Note that both algorithms take the exact same amount of time to finish.

Where is it best to place s? If we want to minimize the total travel, by the formula given above, the best place to place s is the vertex of R that minimizes the sum of distances $\sum_{v\in R} dist(s,v)$ (there may be several). This is the discrete analogue of the so-called Fermat-Toricelli point, or the "geometric median" [KV97].

5.3.2 The number of persistent states

As in previous works on uniform dispersal, our robots are finite-state automatons with $O(1)$ persistent memory bits or states that carry over between time steps. The requirement of finite memory is important, as it allows for scalability: the robots' memory need not scale with the size or complexity of the environment.

There has been some interest in the question of just how little memory one can get away with. It has been shown that oblivious robots - robots with just one persistent state - are incapable of solving the dispersal problem, even with infinite visibility

Algorithm 5.2 5-bit FCDFS

Let v be the current location of A.
if v has no unoccupied neighbours **then**
    Settle.
    $b_3b_4b_5 \leftarrow 011$
else if $b_4b_5 = 00$ **then**
    Search clockwise, starting from the "up" direction, for an unoccupied vertex, and set primary direction to point to that vertex.
    $b_4b_5 \leftarrow 10$
end if
if A cannot move in primary or secondary directions **then**
    if v has just one neighbour **then**
        Settle.
        $b_3b_4b_5 \leftarrow 011$
    else if $(b_5 = 1 \wedge b_3 + b_4 = 1) \vee diag(v)$ is unoccupied **then**
        Settle.
        $b_3b_4b_5 \leftarrow 011$
    else
        Set primary direction to obstacle-less direction not equal to 180° rotation of previous direction stepped in (i.e. the neighbour of v we haven't visited yet; this can be inferred from b_1b_2 and b_3).
        $b_4b_5 \leftarrow 10$
    end if
end if
if b_4b_5 was not updated at this time step **then** ▷ i.e. $b_5 = 1$ or time to update b_5
    $b_4b_5 \leftarrow b_3 1$
end if
if A can move in its primary direction **then**
    Step in the primary direction.
    $b_3 \leftarrow 0$
else if A can step in secondary direction **then**
    Step in the secondary direction.
    $b_3 \leftarrow 1$
else
    Settle.
    $b_3b_4b_5 \leftarrow 011$
end if

[BDS08]. Consequently, any dispersal algorithm requires some number of persistent states, and we are interested in implementing our algorithm with as few as possible - i.e. bringing the robots as close as possible to "obliviousness" of their prior history and to center their decisions, as much as possible, on their current position and frame of reference.

Moreover, Algorithm 5.1 required the robots to remember their previous locations relative to their current location and to be able to use them as points of comparison. The 5-bit implementation shows how this could be done through remembering only the previous two relative directions of motion. A robot is then required only to know

whether there are obstacles at the four cardinal directions (up, down, left, right), and at its diagonal, which is always at a 135° degree rotation from the primary direction. This simplifies the localization computations.

We implemented a 5-bit or 2^5-state version of our algorithm on a simulator (see Algorithm 5.2). A robot's state is described by bits $b_1b_2b_3b_4b_5$. All bits are initially 0. b_1b_2 describe the primary direction (one of four), and b_3 tells us whether the previous step was taken in the primary direction (if $b_3 = 0$) or in the secondary direction (if $b_3 = 1$). b_4b_5 is a counter that is reset to 10 upon entering a hall or one step after initialization, and thereafter is equal to $*1$, where $*$ is a bit that tells us whether we walked in the primary or secondary direction two steps ago (by copying b_3). A robot that detects an obstacle at its diagonal interprets its position as a fake hall (i.e. a corner) as long as $b_5 = 1$ and $b_3 + b_4 = 1$, that is, as long as at least one time step passed since the last hall, and our previous position was diagonal to us. In order to conserve memory, our robots do not strictly speaking have a "settled" state. Instead, once a robot determines it is in a corner (and so needs to settle), it sets $b_3b_4b_5$ to 011, indicating that it visited its diagonal–this causes it to never move again.

5.3.3 The impossibility of minimizing total travel for general grid environments

We saw that there is a local rule that minimizes total travel for simply connected grid environments. In this section we show that, for robots with finite visibility, there is no local rule that universally minimizes total travel for all connected grid environments.

Let r be the visibility range of the robots. Consider the grid environment in Figure 5.4 (not drawn to scale). It connects a set of $10r$ columns of width 1 spaced $2r$ cells apart. The bottom row has total length $20r^2$. Most of the columns are dead-ends and have a height of $30r^2$. The first column and an additional column connect to the top row, and have height $30r^2 + 1$. Label the grid environment where this additional column is the kth column $G(k)$. The door s is at the bottom left.

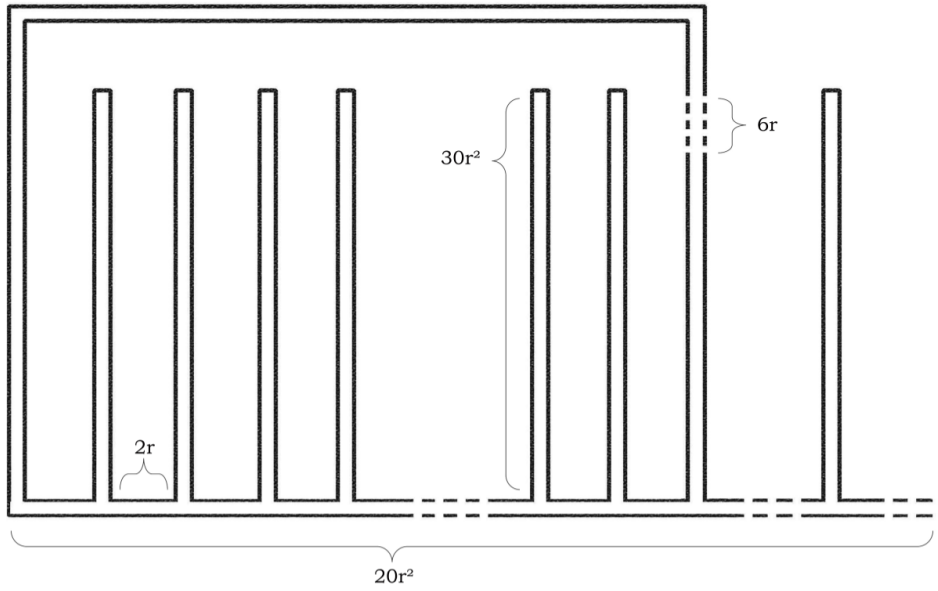
It is readily seen that the total travel required by an optimal solution for any environment $G(k)$ is $\sum_{v \in G(k)} dist(s, v)$, where s is the door of $G(k)$ (let a line of robots going up the first column fill the top row, and let robots going to the right fill the other columns).

Proposition 5.3.9. *Let **ALG** be a local rule for uniform dispersal of robots with visibility range r. There is an environment $G(k)$ for which the total travel of **ALG** is at least $\sum_{v \in R} dist(s, v) + 1$.*

Proof (Sketch) We consider the actions of rule **ALG** on the grid environment $G(k)$. We do not specify the value of k yet.

As before, label the robots emerging at s $A_1, A_2, \ldots$ in their order of arrival. Since A_1 cannot distinguish between the up and right directions upon arrival at s (any distinct

Figure 5.4 The construction $G(k)$.



feature of the environment is at distance at least $r + 1$ and hence is invisible), we can assume without loss of generality that it steps up (if it steps right, simply rotate and reflect $G(k)$).

Assume for contradiction that the total travel of **ALG** is $T = \sum_{v \in G(k)} dist(s, v)$. This assumption implies that every robot travels a shortest path to its settlement destination. In particular, A_1 must have precisely $dist(A_1, v_1)$ travel, where v_1 is the destination at which A_1 chooses to settle.

We note the following facts:

1. Once A_1 stepped up, it has committed to stepping up and right until reaching v_1, as circling in place or going in a third direction increases its travel past $dist(A_1, v_1)$, causing the total travel of **ALG** to be greater than T–a contradiction.

2. v_1 cannot be a vertex in the first column or in the top row except the top vertex of column k or one vertex to its left, as should v_1 not equal those, settling there would block off the path to the top row going through the first column, and force other robots to travel to the top row through column k. This is sub-optimal, and causes the total travel to increase beyond T–a contradiction.

3. v_1 cannot be any vertex in the kth column other than the top of the kth column, as this would require A_1 to step downwards.

(*) From (1)-(3) we conclude that v_1 must equal precisely the top vertex of the kth column or one vertex to its left.

Up to the time when A_1 reaches the top row, none of the ends of the other columns have been seen, so **ALG** will run the same regardless of the value of k. Since total

travel is assumed to be optimal, no robot can block s for more than one time step, so by the time A_1 reaches the top row, there will have been created at least $4r$ robots. Each of these $4r$ robots must have already entered one of the columns or settled, since they travel optimal paths to their destination, and the total length of the bottom row is $20r^2$, whereas $30r^2$ time must have passed for A_1 to reach the top.

As there are $10r$ columns, there must exist a column that none of the robots $A_1, \ldots, A_{4r}$ have entered. Set the value of k to equal this column.

When A_1 reaches v_1, the above indicates that any other robot currently present in the kth column (if there are any) arrived at least $2 \cdot 4r$ time steps after A_1. Therefore it is at distance at least $8r$ from A_1, meaning that there is a space of $6r$ vertices in column k that no robot has seen yet. This indicates that **ALG** must make the same decision for A_1 whether these vertices exist or not. However, if any one of these vertices does not exist, then column k is not connected to the top row, indicating that A_1 cannot settle at the top of the kth column or to its left, else it will block off part of the environment. We arrived at a contradiction to (*).

We conclude that there is an environment $G(k)$ where the total travel of **ALG** is greater than the optimum, so ALG is sub-optimal.

By adding more columns to the $G(k)$ construction and increasing the height of the columns, we can force A_1 to go down more and more steps, causing the difference between the optimal total travel and the total travel of **ALG** to be arbitrarily large.

Proposition 5.3.9 only makes the assumption of limited visibility. It holds even assuming the agents have global communication, infinite memory, and are aware of each others' positions at all times.

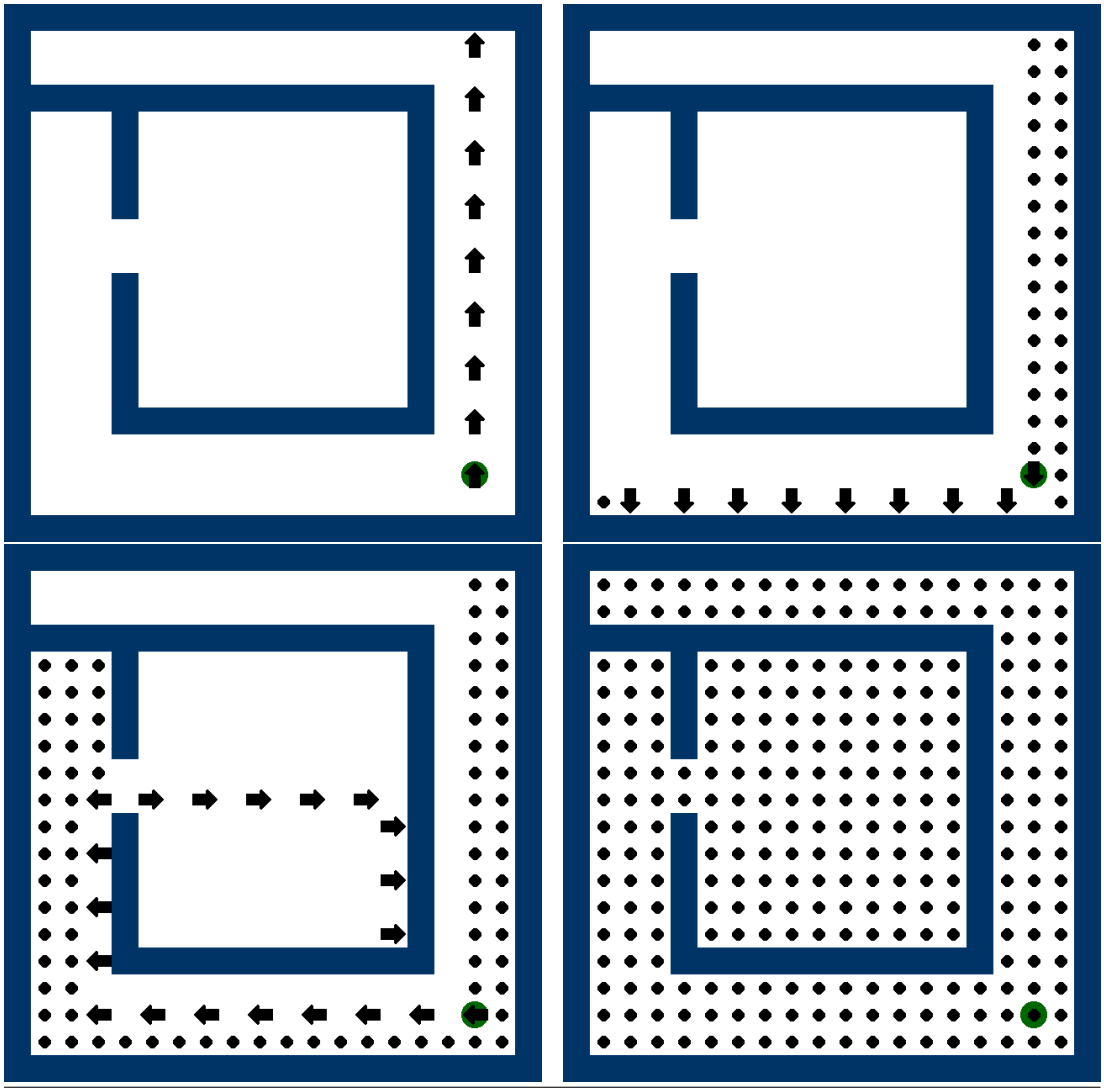
We note that we did not exclude the possibility of a local rule that minimizes the maximal individual travel. Furthermore, we did not exclude the possibility of a rule that minimizes total travel when pauses are not counted.

5.4 Simulations, comparisons, and alternative strategies

We verified and animated our algorithm by simulating it on our robot simulator. Figures 5.5 and 5.6 show four stills from a run of the algorithm on two different environments. Figure 5.7 shows a FCDFS deadlock scenario in an environment that is not simply connected: the halls constantly redirect the robots, forming a cycle. The door vertex has mistakenly blocked itself off, due to the robots exiting from it mistaking the robots in a cycle for obstacles.

We experimented with two variants of FCDFS that are similarly optimal. FCDFS assumes robots are initialized with a common notion of up, down, left and right, but this assumption is unnecessary if we let robots settle in place as soon as they reach a corner (in FCDFS they keep moving if they can). This modified strategy is illustrated in Figure 5.8, where robots randomly choose their initial direction. This creates a more

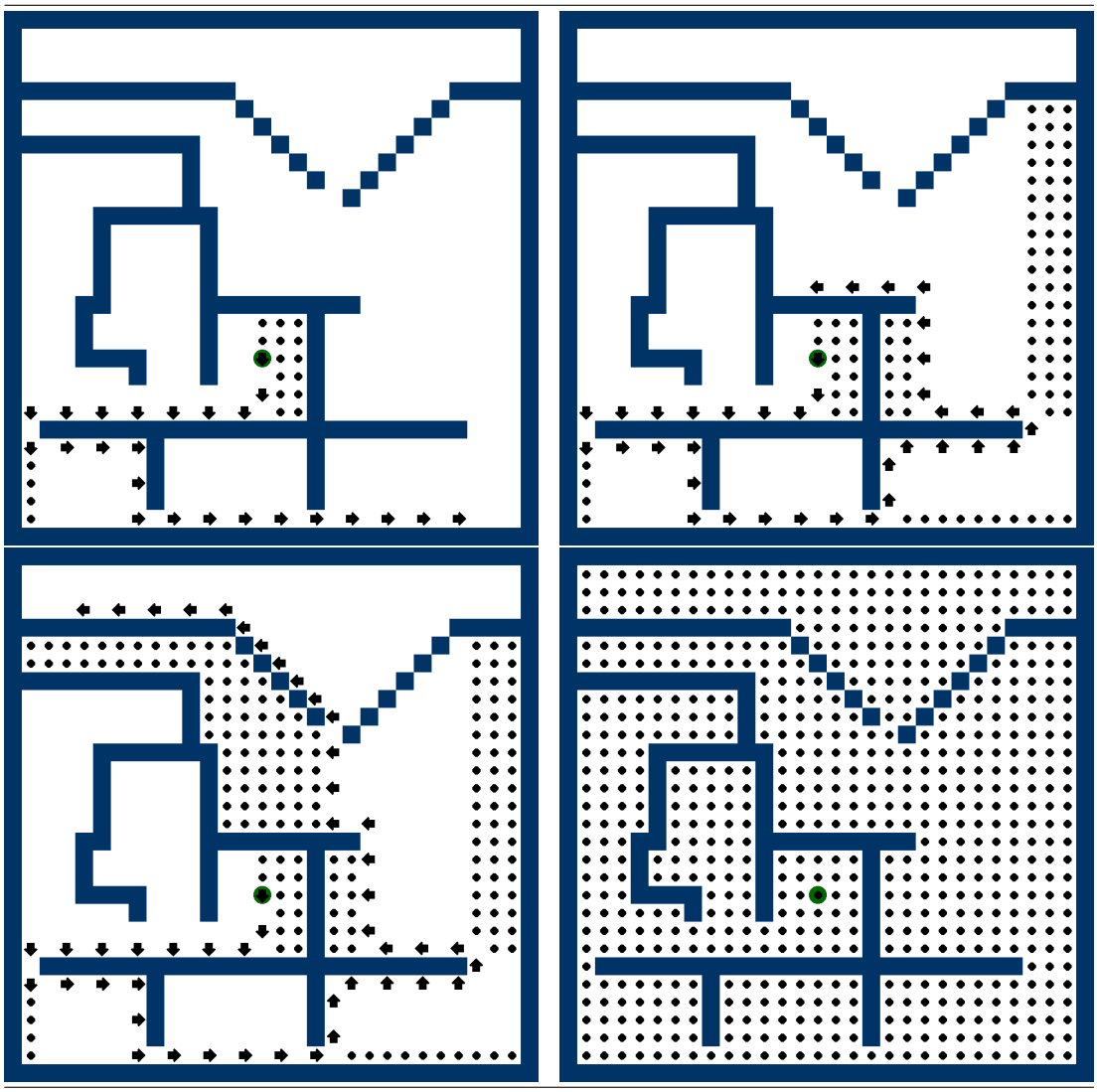
Figure 5.5 A simulation of FCDFS. The blue blocks are walls. The arrows indicate the location and primary direction of the robots, and the diamonds are settled robots. Rather than block active robots, the settled robots form halls to enable the swarm to explore more of the environment.



"symmetric"-looking dispersal. The strategy shown in Figure 5.9 is more significantly different: in it, rather than stick to their secondary and primary directions, robots attempt to scale the boundary of the environment with a "left hand on wall" clockwise orientation, until they hit a corner or a wall. Both of these variants achieved the same makespan and total travel as FCDFS, though they are visually distinct.

Empirically, we compared the performance of FCDFS to the performance of our implementation of the DFLF and BFLF algorithms of [HAB+04] (adapted to our slightly different model) over a number of simply-connected environments, measuring the total travel and maximal individual travel (Table 5.1). Note that though all algorithms are deterministic, some local decisions are not fully specified in [HAB+04], hence different implementations may result in slightly different performance, though asymptotically

Figure 5.6 A simulation on a different environment. Note how the trail of robots always forms a shortest path to its current front.



every implementation will perform the same. We let our robots decide between arbitrary local decisions at random, averaging performance over several re-runs.

Only for the sake of this comparison, we elected to exclude time steps where robots are active but do not change location, as such intermediate pauses are not counted in [HAB+04]. FCDFS is optimal regardless, and factoring these in leaves the DFLF and FCDFS columns unchanged, since such pauses never occur during their execution. However, including pauses causes the *maximal travel* of BFLF to become extremely large. Hence, Table 5.1 shows that BFLF is good at reducing the number of location changes of a robot, but in many applications (e.g. when robots are quadcopters) its *energy* consumption is very high compared to FCDFS.

Figure 5.7 A deadlock scenario in environments that are not simply connected.

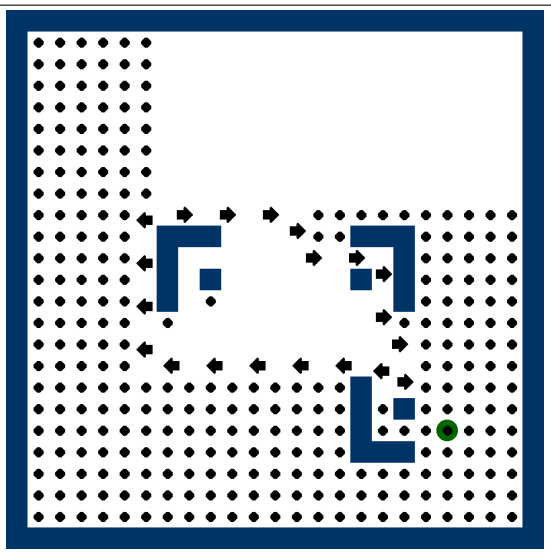
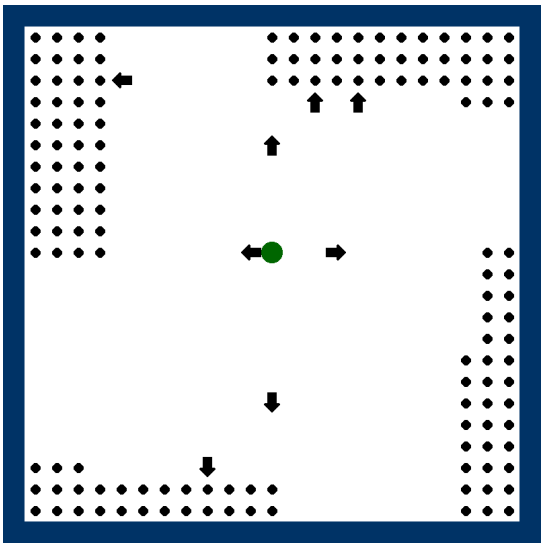


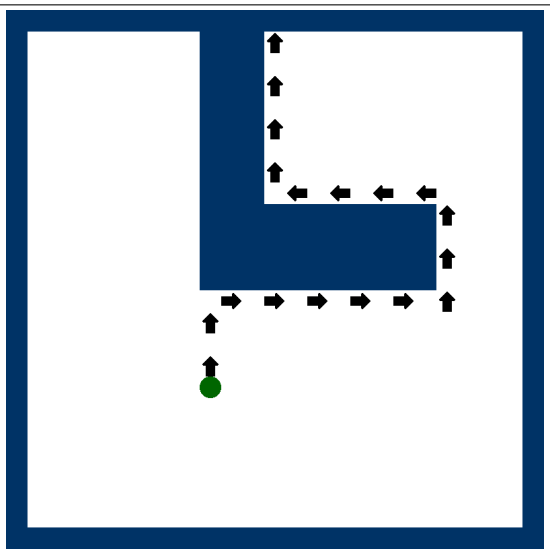
Figure 5.8 Multi-directional dispersal strategy.



	DFLF	BFLF (excl. stops)	FCDFS
30x30 Grid	237984 (460)	16323 (50)	13620 (32)
Fig. 5 Environment	16139 (126)	6742 (50)	5909 (38)
Fig. 6 Environment	100419 (296)	39576 (112)	35103 (99)
Fig. 9 Environment	50889 (190)	7283 (39)	6600 (35)

Table 5.1: A comparison of total travel and maximal individual travel over different environments (excluding pauses). Entries are in the form *total travel (maximal travel)*. See Figures 5, 6, 9 for the specific environments used.

Figure 5.9 "Left hand on wall" strategy.



5.5 Discussion

A robotic swarm must take into account the energy capacity of the individual. We discussed the problem of minimizing travel, hence energy expenditure, in the uniform dispersal problem for simply connected grid regions. We showed the existence of a local strategy that minimizes total and individual travel for the case of a single source vertex. We showed also a non-existence result for such strategies in the case of general grid environments.

Several extensions of our work can readily be considered. First, as our algorithm deals only with the single door case, it is desirable to find an energy-efficient dispersal algorithm for the case of multiple doors from which robots arrive independently.

Next, synchronicity is a strong assumption, enabling every robot to proceed to its destination without ever being blocked by another robot. To extend our work to less controlled settings, we may assume an asynchronous time scheme–for example, allow a probability q that an agent fails to activate at a given time step. We cannot expect a makespan- and total travel-*optimal* algorithm to exist in such settings, but we anticipate relatively effective strategies might exist. As a way to proceed, though our algorithm makes the powerful assumption of synchronicity, the strategy of finding corners and not stopping at halls seems general, and could possibly be adopted for the asynchronous case as well.

Finally, our algorithm requires the environments to be simply-connected orthogonal environments: it would be interesting to see an algorithm that works for broader scenarios, or in the opposite direction, results regarding the non-existence of efficient algorithms for such scenarios under the stringent computational assumptions we made.

Chapter 6

Swarm Robotics II: Uniform Dispersion With Crash-prone Robots

In Chapter 5 we described an ant-like energy and time-optimal algorithm for swarm uniform dispersion over simply connected grid environments. This algorithm operated under two idealized assumptions: (i) robots activate synchronously in each time step and (ii) robots are not subject to memory errors or crash faults. In the current chapter, which is based on the paper [AB20], we revisit the uniform dispersal problem under a less forgiving set of assumptions. Our goal is to find an algorithm that can complete uniform dispersal in a reasonable time span (but without necessarily minimizing energy) assuming robots activate asynchronously and can suddenly crash and disappear from the environment. Additionally, unlike Chapter 5 where we assumed the environment was a simply connected grid graph, here we assume the environment the robots traverse is represented by an arbitrary, a priori unknown graph G.

Our proposed solution to both the problem of asynchronous coordination and of crashing is a *dual role* algorithm wherein each mobile robot is both an explorer and, once it settles somewhere, a *stationary node* that helps other agents navigate the region. This idea draws upon the bio-inspired concept of *stigmergy*–communication via the environment [KES01]. In nature, social insects often communicate by leaving pheromones or other kinds of marks inside the environment. Our implementation of stigmergy utilizes the agents themselves as a medium to change the state of the environment, in the form of agent nodes that after settling become a part of the environment. Our dual role approach enables robots which are displaced (e.g., due to strong winds or long stretches of time where they are not activated) to quickly find their bearing and resume executing their algorithm, and also enables the swarm to be resilient to crashes in the mobile agent layer.

The benefits of agent-node duality in swarm-robotic coverage tasks are not limited

to asynchronicity and crash mitigation. Recalling Chapter 5, we would be remiss not to mention our joint work with Ori Rappel, [RAB22] (the full version of which is not included in this dissertation), where we build upon the concept of two- layer swarms to create highly energy efficient algorithms for uniform dispersion by equipping stationary agent nodes with the ability to send back-propagating signals to other agent nodes.

Such ideas are outside our current scope: in this chapter, we focus on a highly straightforward, DFS-esque uniform dispersal algorithm which can be described in 9 lines of code (Algorithm 6.1). Despite the algorithm's simplicity, its rigorous analysis turns out to be fairly subtle. The analysis is crucially reliant on a lengthy reduction to the totally asymmetric simple exclusion process (see the Preliminaries) and non-elementary results from the TASEP literature. We would be highly interested in a more elementary (but still rigorous) analysis and pose this as an open problem to any interested reader.

6.1 Introduction

Swarms are often claimed to be highly fault-tolerant, as redundancy and sheer numbers can enable the swarm to go on with its mission even if many robots malfunction [WN06]. However, as the size of a robotic fleet grows, so too does the opportunity for error. Specifically, three different complications that arise in multi-robot systems are further exacerbated in the swarm setting:

Asynchronicity. As the number of robots grows, coordinating the robots' actions becomes a formidable task, as their actions and internal clocks can become highly unsynchronized.

Crashes. We cannot expect to release a huge swarm of simple robots to an unknown environment without the occurrence of hardware or software faults that may cause robots to crash.

Traffic. To avoid collisions, we do not wish for there to be too many robots crowding a given area, and so mobile robots should maintain safe distances from each other. In restricted physical environments, such requirements cause traffic delays, as robots must wait for other robots to move away before entering a target location.

Such challenges are discussed as a central direction of research for swarm robotics in [Pel05]. If the number of errors scales with the number of robots, are swarms "worth the trouble"? The purpose of this chapter is to give a perspective on this question via a formal mathematical analysis. We study, in an abstract setting, the ability of a simple local rule to achieve *uniform dispersal* in the presence of crashes and asynchronicity. We are specifically interested in how the frequency of crashes affects the time to mission completion.

We first describe a rule of behaviour for swarms that is capable of achieving uniform dispersal which uses two types of agents, both of which can be present in the same location. Using this algorithm, we show that a swarm can complete its mission

quickly and reliably in a priori unknown discrete environments, even in the presence of asynchronicity and frequent crashes. Hence, we claim that in our setting, many robots can win against many errors. In the spirit of swarm robotics, the algorithm relies only on local information to dictate robots' actions.

Our swarm consists of a large reservoir of simple, anonymous, identical, and autonomous mobile robots that enter the environment over time via a source location s. The robots move across a discrete environment represented by an *a priori unknown* graph G whose vertices represent spatial locations. The robots gradually expand their coverage of the environment by occupying certain locations and assisting other nearby robots in navigational tasks using a local, indirect communication scheme.

The swarm's robots have two modes: mobile and settled. The settled robots act as "nodes" or "beacons" of the current coverage of the graph environment, and the mobile robots move between locations with settled robots until they can find a new location where they themselves can settle. The settled 'robot nodes' are capable of *pointing to* ("*marking*") a single neighbouring location where there is another settled robot. "Marking" is understood to be a generic capability of the robots and could be accomplished by many different technologies, such as local radio communication or visual sensing; we refer to the Related Work section for possible implementations.

As more and more mobile robots become settled, their marks serve as a navigational network of the environment that is utilised by the remaining mobile robots. The mobile robots are capable of sensing the number of robots in neighbour locations, and sensing when a settled robot is pointing to (marking) their location. They rely only on this information to make decisions. Hence, they operate in an ant-like, GPS-denied, low memory setting, meaning they act based only on local communications and local geographic features. The robots are tasked with settling at every vertex of G, and constructing an implicit *spanning tree* of G via the settled robots and their pointer marks.

There are no restrictions on G as long as it is connected. In principle, different robots need not even agree on the graph representation of their environment for our algorithm to work (e.g., in case they gradually build it from local sensory data), as the settled robots gradually construct a spanning tree which all robots agree on and use to move between locations. We assume, for simplicity, that they share the same representation.

Physical constraints and asynchronicity. We model the mobile robots as activating repeatedly at stochastic, independent exponential waiting times of rate 1. When a robot activates, it may move or move-and-settle at a nearby location (once a robot settles, it remains stationary). We assume the physical constraint that any given location may contain no more than a single mobile robot and a single settled robot. Due to asynchronous activation times, frequent traffic obstructions occur as robots block each other off from progressing.

This model of asynchronicity and limited vertex capacity in a graph environment

is motivated by the *totally asymmetric simple exclusion process* (TASEP) in statistical mechanics. There is an extensive literature on this process as a model for a great variety of transport phenomena, such as traffic flow [CSS00] and biological transport [CMZ11]. Rigorous exact and asymptotic results for TASEP are known [Joh00; TW09], and our analysis technique shall be to compare our swarm's performance to a two-layered TASEP-like process. Since our robots are mostly in a state of "traffic flow" (waiting for other robots to move), references such as [CSS00] suggest that our model, despite being inherently idealized, in fact captures many of the relevant traffic phenomena that will occur in real life implementations.

Adversarial crashing. Similar to, e.g., [JCMP17], we consider a risky traversal model where robots may crash whenever they try to move across an edge. We assume robots remain safe when not moving, as remaining put is less risky than travelling (in fact, we need just the weaker assumption that settled robots, which *never* move, are safe). To facilitate analysis, we assume crashed robots disappear from the environment. This assumption is applicable when such robots can be manoeuvred around or pushed aside, or we may consider crashed air-based robots falling to the ground during exploration of an environment. Alternatively, in a ground robot scenario, we can with foresight expand vertex sizes to be big enough such that vertices can contain a small number of crashed robots in addition to the two active robots (and such local crashed robots are then bypassed using, e.g., local collision avoidance).

We assume that the number of crashes that occur is bounded by the current time t, and a parameter c which reflects the frequency at which crashes occur over time. When c is close to 1, the *vast majority* of robots that enter the environment will crash before achieving anything.

Besides these limitations, we assume nothing more about the crashes that occur. In particular, a *virtual adversary* may choose crashes so as to be as obstructing as possible.

Results. We describe a local rule of behaviour (Algorithm 6.1) that can achieve uniform dispersion, even in the presence frequent crashes and traffic obstructions. The rule is easy to understand and implement and is well-suited for a swarm of simple robots, mimicking a kind of branching depth-first search. In many mobile robot systems one wishes to construct a spanning tree of the environment for purposes of mapping, routing or broadcasting [AMZ06; AHK06; Bro89; DP12; GR01]. Our rule achieves this as well, by having robots act as nodes of the tree, and making them aware of their immediate descendants. Our goal is to study how crashes, asynchronicity, and traffic affect the swarm's performance under this rule of behaviour.

We prove that our robots are able to complete their mission in time linear in the size of the environment, and that performance degrades gracefully (by a factor $(1-c)^{-1}$) with frequency of crashes. Given our assumptions and algorithm, it is not surprising that the robots can complete the dispersion assuming some crashes; rather, we show that even with many frequent crashes, the robots can still do so efficiently.

Specifically, let n be the number of vertices in the environment G. We prove that

dispersal completes before time $8 \cdot ((1-c)^{-1} + o(1))n$ asymptotically almost surely (meaning with probability approaching 1 as n grows)–a worst-case bound on performance. No dispersal algorithm can complete in less than $O(n)$ expected time, since this is the time it takes to even explore n vertices, so when there are no crashes (but still there is traffic and asynchronicity) this bound is asymptotically tight. For, say, $c = 0.5$, we expect up to (roughly) 50% of robots to crash before achieving anything, and our analysis says that therefore the swarm will take twice as long to achieve dispersal. This seems intuitive, but consider that the robots that eventually crash are (uselessly) present in the environment in the time leading to the crash, blocking other robots from entering or progressing. The analysis says that nevertheless, the ability of the rest of the swarm to achieve its goal is not disproportionately worsened.

To the best of our knowledge, with or without crashes, we are the first to consider a non-synchronous setting for the uniform dispersal problem where time to completion can explicitly be bounded, hence also the first to give explicit performance guarantees in a non-synchronous setting. In an asynchronous as opposed to a synchronous setting, there are many more possible configurations that the robots might exist in, which makes the analysis more difficult. We believe the TASEP references and techniques from statistics [Joh00; TW09; CSS00] might be of general interest for tackling these kinds of topics.

Our analysis extends also to a synchronous time setting, and to the case where robots enter the environment from *multiple* locations. Multiple entrance locations result instead in the robots constructing instead an implicit *spanning forest*. In both these settings, dispersion completes faster. The bound on performance we derive for the synchronous case is exact.

Finally, we confirm our findings by numerically simulating our system in a number of environments and measuring performance.

6.1.1 Related work

As mentioned in the previous chapter, uniform dispersal was introduced by Hsiang et al. in [HAB+04] for discrete grid environments of connected pixels (but their work can be extended to arbitrary graph environments). They considered a synchronous time setting where robots are allowed to send short messages to nearby robots, and showed time-optimal algorithms for this setting. Many variations have since been studied. Barrameda et al. extend the problem to the asynchronous setting with no explicit non-visual communication [BDS13; BDS08]. Recent works include dispersal with weakened sensing [HL17a], dispersal in arbitrary graph environments [KA19]. Our model differs from previous work in several points, including the presence of crashes, the two layers, and the ability to mark neighbours. Marking is weaker than the radio communication available to robots in [HAB+04] that enables robots to transfer many bits of data locally, but stronger than the indirect, visual communication assumed in some other

Reference	Environment type	Time setting	Communication	Makespan	Crashes
[HAB+04]	Arbitrary	Synch.	Radio	$O(n)$	x
[AB19b]	Simply connected grid	Synch.	Visual	$O(n)$	x
[HL17a]	Grid	Synch.	Visual	$O(n)$	x
[BDS08]	Simply connected grid	Asynch.	Visual	undefined	x
[BDS13]	Grid	Asynch.	Radio	undefined	x
This chapter	Arbitrary	Stochastic Asynch.	Marking	$O(n)$	✓

Table 6.1: A comparison of works on uniform dispersal.

works.

Because of differences in the settings, assumptions, and constraints, quantitative comparison of works on uniform dispersal is very difficult. Table 6.1 gives a rough, non-exhaustive overview of some differences, such as the supported kinds of environments (grid environment, hole-less grid environment, or arbitrary graph environment), synchronous versus asynchronous time, expected makespan (i.e., how long it takes the robots to complete their mission), and whether crashes are considered in the model.

Robotic coverage, patrolling, and exploration with adversarial interference, as well as crashes, have been studied in different problem settings from our own. Agmon and Peleg studied a gathering problem for robots where a single robot may crash [AP06], and gathering with multiple crashes was later discussed by Zohir et al. in a similar setting [BDT13]. Robotic exploration in an environment containing threats has been studied in [YAK13; YAK15]. Moreover, adversarial crashes of processes are often studied in general distributed algorithms (e.g., [DFGT11]). Differing from many of these works, we study a situation where the number of crashes scales with the mission's complexity (the time it takes to cover the environment), and where even the vast majority of robots may crash. However, to enable this, we assume access to a huge reservoir of robots waiting to replace crashed robots–i.e., a robotic swarm.

A fascinating introduction to TASEP-like processes and their connection to other fields is [KK10].

6.2 Model and System

We consider a swarm of mobile robotic agents performing world-embedded calculations on an unknown discrete environment represented by a connected graph G. The vertices of G represent spatial locations, and the edges represent connections between these locations, such that the existence of an edge (u, v) indicates that a robot may move from u to v.

We assume an infinite collection of robots (also referred to as 'agents') attempt to enter G over time through a *source* vertex $s \in G$. The robots are identical and execute the same algorithm. They begin in the *mobile* state, and eventually enter the *settled* state. Settled robots are stationary, and are capable of *marking* a neighbouring vertex

that contains another settled robot. Mobile robots move between the vertices of G and sometimes crash while in motion. They are oblivious and ant-like, deciding where to move based only on local information provided by their sensors, i.e., the number of robots at neighbouring vertices, and whether any of the neighbouring settled robots mark their current location. Each vertex has limited capacity: it can contain at most one settled and one mobile robot.

Mobile robots are only allowed to move to a neighbouring vertex when they are *activated*. Each robot, including robots outside G, reactivates infinitely often and independent of other robots, at random exponential waiting times of mean 1.

When s contains less than two robots, robots from outside G attempt to enter it when they are activated. It is convenient to give the robots arbitrary labels $A_1, A_2, \ldots$ and assume that A_i cannot enter s before all robots with lower indices entered or crashed. This assumption makes the analysis simpler, but the performance bound we prove in this chapter holds also for the entrance model where robot entrance depends only on which robot is activated first. Hence, whenever the current lowest-index robot outside of G activates and there is no *mobile* robot at s, it moves to s. If s is completely empty, the robot settles upon arrival and becomes the root of the spanning tree. Otherwise it remains a mobile robot.

We denote by $G(t)$ the graph whose vertices are vertices of G containing settled robots at time t, and there is a directed edge $(u, v) \in G(t)$ if u is marked by a settled robot at v. The goal of the robots is to reach a time T wherein $G(T)$ is a spanning tree of the entire environment G. The *makespan* of an algorithm is the first time T_0 when this occurs.

Crashes are modelled as follows: when a robot A_i is activated and attempts to enter s or move from u to v via the edge (u, v), occasionally an *adversarial event* occurs, causing the deletion of A_i from G. Robots do not crash unless attempting to move. Hence, mobile robots are volatile but settled robots are safe. This assumption is somewhat stronger than necessary: our results still hold if mobile (but not settled) robots are allowed to crash while they stay put, but this tediously lengthens the analysis. We assume the number of adversarial events before time t is bounded by $ct/4$ where $0 \leq c < 1$ is some constant. Adversarial events may otherwise be as inconvenient as possible: we may assume there is an *adversary* choosing crashes to maximize the makespan of our algorithm.

Unless stated otherwise, when discussing the configuration of robots "at time t", we always refer to the configuration before any activation at time t has occurred.

6.3 Dispersal and Spanning Trees

We study a simple local behaviour (Algorithm 6.1) that disperses robots and incrementally constructs a distributed spanning tree of G. The rule determines the behaviour of *mobile* robots whenever they are activated (settled robots merely remain in place and

continue to mark their target). We prove that using this rule, the makespan is linear in the number of vertices of G asymptotically almost surely, and that performance degrades gracefully with the density of crashes.

Algorithm 6.1 Local rule for a mobile robot A.

Let v be the current location of A in G (if A is outside G, see Section 6.2).
if a neighbour u of v contains exactly one robot, and this robot marks v **then**
    Attempt to move to u.
else if a neighbour u of v contains no robots **then**
    Attempt to move to u and become *settled* if no crash.
    Mark the vertex v.
else
    Stay put.
end if

The rule grows $G(t)$ as a partial spanning tree of G. It acts as a kind of depth first search that splits into parallel processes whenever a mobile robot is blocked by another mobile robot. Every vertex of the tree $G(t)$ is marked by settled robots at its descendants. Mobile robots follow these marks to discover the leaves of the current tree $G(t)$ and expand it. Robots grow the tree by settling at unexplored vertices that then become new leaves. Our main result is Theorem 6.1:

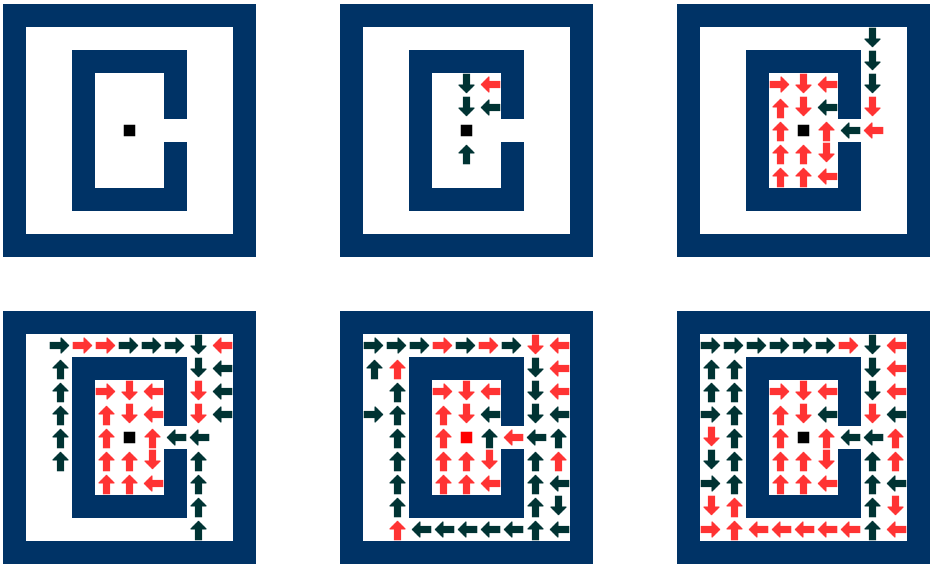
Theorem 6.1. *If for all t the number of adversarial events before time t is allowed to be at most $ct/4$, $0 \leq c < 1$, then the makespan of Algorithm 6.1 over graph environments with n vertices is at most $8((1-c)^{-1} + o(1))n$ asymptotically almost surely as $n \to \infty$.*

Figure 6.1 shows an execution of our algorithm on a grid environment with $n = 62$ square vertices (white region) and obstacles (blue region). We allowed a naive adversary to arbitrarily delete at most $ct/4$ robots before time t, with $c = 0.8$. This corresponded to a deletion of 56% of robots that entered the environment before the makespan. In a more constrained topology (such as a path graph, see Section 6.3.1), the robots would progress more slowly, and a greater percentage would be deleted. The makespan (bottom right figure) was 613, consistent with the upper bound of Theorem 6.1. After the construction of the spanning tree by settled agents completes, robots keep entering the region until there are two robots at every vertex. This is related to the "slow makespan", which we will later define. The slow makespan was 831. See Section 6.4 for more simulations.

6.3.1 Analysis

We study the makespan of Algorithm 6.1. For the sake of flow, some of the more technical proofs have been moved to the "Analysis details" section (Section 6.5).

Figure 6.1 An execution of Algorithm 6.1 on a grid environment. The source is denoted by a square box in the center. The arrows denote settled robots, and their direction points to the adjacent location with a settled robot that they mark. Red arrows indicate a mobile robot is on top of the settled robot (note that by the algorithm, a mobile robot will never occupy a vertex that does not have a settled robot). The environment is a priori unknown to the robots, and they construct a spanning tree representation of it over time.



For the analysis, we will assume that robots from $A_1, A_2, \ldots$ that settle or crash keep being activated. This is a purely "virtual" activation: such robots of course do and affect nothing upon being activated. We start with a structural Lemma:

Lemma 6.3.1. *$G(t)$ is a tree at all times t with probability* 1.

Proof When the first robot enters and successfully settles, $G(t)$ contains only s. No settled robots are ever deleted, so $G(t)$ can only gain new vertices. Whenever a mobile robot settles, it extends the tree $G(t)$ by one vertex, connecting its current location v to $G(t)$ via a single directed edge. By definition, the edge is directed from the vertex the settled robot *marks*–which is its previous location–to v. This turns v into a leaf of $G(t)$. With probability 1 no two robots on G activate at the exact same time, so no two robots settle the same vertex. Hence $G(t)$ remains a tree. ■

Event orders

We explain how we intend to bound the makespan. Our strategy shall be to use coupling to compare the performance of Algorithm 6.1 by the performance of different random processes of robots moving on different structures.

The basic idea is this: whenever we run Algorithm 6.1 on G, we can log the exact times at which the robots activate, as well as the times adversarial events happen and which robots they affect. This gives us an *order of events* S sampled from some

random distribution. Note that robots keep activating forever (but these activations do nothing once the graph is full of robots), so S is infinitely long. We then "re-enact" or **"simulate"** S on a new environment (or several new environments) involving the robots $A_1, A_2, \ldots$ by activating and deleting the robots according to S.

To make things more precise, by "simulating" S on different environments we mean that we consider the *coupled* process $(G, G_2, \ldots, G_m)$ wherein different environments $G, G_2, \ldots G_m$ have robots that are *paired* such that whenever A_i in G is scheduled for an activation or a deletion according to the event order S (S is simply an infinite list of scheduled activation and deletion times), the copies of A_i in *all* the environments $G_2, \ldots G_m$ also activate or are deleted. When the copies of A_i are activated they act according to Algorithm 6.1 with respect to their local neighborhood. Robots entrances are modelled as usual (Section 6.2), but note that even if A_i manages to enter G following an activation, its copy might not enter its own environment because in that environment the entrance is blocked, or there is a lower-index robot waiting to enter. During Algorithm 6.1's analysis, we will often be talking about a deterministic event order S being simulated over different environments. The end-goal, however, is to say something about the event order S when it is randomly sampled from the execution of Algorithm 6.1 on G.

The event order S must be a *possible* set of events that occurred during an execution of our algorithm on the base graph environment G. This means, due to our model, that a robot A_i in G will never be scheduled for deletion except at times when it is activated and attempts to move. However, while simulating S on the environments $G_2, \ldots G_m$, we must be allowed to break the rules of the model: we might delete robots even when they don't attempt to move, or while they are outside of the new graph environment. Whenever we say "for any event order S", we mean event orders S **that could have happened over** G.

In S, define t_0 to be the first time A_1 activates, t_1 to be the first time *after* t_0 that either A_1 or A_2 activate, and t_i to be the first time $t > t_{i-1}$ that any robot in the set $\{A_1, \ldots, A_{i+1}\}$ is activated.

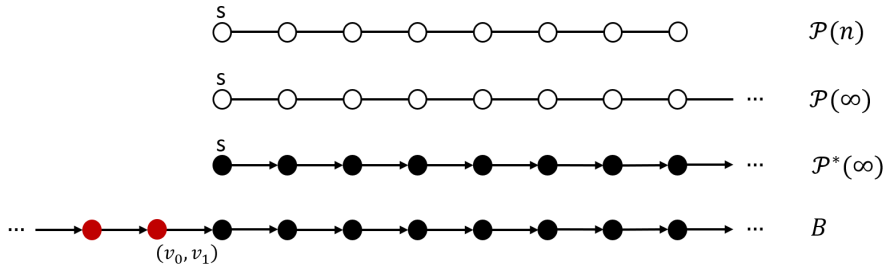
Definition 6.3.2. The times $t_0 < t_1 < t_2 < \ldots$ in S are called the **meaningful event times** of S.

For meaningful event times to be well-defined there must be a *minimal* time $t > t_{i-1}$ where one of the robots $A_1, \ldots, A_i$ activates. Because the activation times of the robots are independent exponential waiting times of mean 1, this is true with probability 1 for a randomly sampled S. Moreover, with probability 1, at any time t_i there is precisely one robot A of $A_1, A_2, \ldots, A_{i+1}$ scheduled for activation by S. Because both these things are true with probability 1, we assume they are true for any event order S referred to at any point in this analysis. This does not affect our main result (Theorem 6.1), which is probabilistic.

Our end-goal is randomly sample S from G and simulate it on four increasingly "slower" environments: $\mathcal{P}(n)$, $\mathcal{P}(\infty)$, $\mathcal{P}^*(\infty)$, B, so that all environments (G and these four) are coupled. *Meaningful event times* are so called because, prior to the first activation of A_i, any of the robots $A_{i+1}, A_{i+2}, \ldots$ cannot enter or move in any of these environments, and activating them causes nothing. Hence, at any time t which is not a meaningful event time, the configuration of robots cannot change (no robots move and no robots are deleted in any of the environments S is simulated on).

The possibility to create an event order S is the only reason we labelled the robots and made the assumption about entrance orders in Section 6.2; the robots themselves are not aware of their labels and make no use of them.

Figure 6.2 The processes $\mathcal{P}(n), \mathcal{P}(\infty), \mathcal{P}^*(\infty), B$ that we will be interested in. White vertices are empty, black vertices contain a settled robot, and red vertices contain both a mobile and a settled robot. Edge directions indicate edge directions in $G(t)$. Note that B does not have a source vertex.



$\mathcal{P}(n)$ versus G

Let n be the number of vertices of G. The path graph $\mathcal{P}(n)$ over n vertices is a graph over the vertices $v_1 v_2 \ldots v_n$ such that there is an edge (v_i, v_{i+1}) for all $1 \leq i \leq n-1$. We simulate S on the graph environment $\mathcal{P}(n)$ where the source vertex s is v_1. Simulating S on $\mathcal{P}(n)$ results in what is mostly a normal-looking execution of Algorithm 6.1 on $\mathcal{P}(n)$, but as discussed, it might lead to some oddities such as robots being deleted while they are still outside the graph environment.

Let us introduce some notation. A_i^G refers to the copy of A_i being simulated by S on G, and $A_i^{\mathcal{P}(n)}$ is similarly defined.

Definition 6.3.3. The *depth* of A_i^G at time t, written $d(A_i^G, t)$, is the number of times A_i^G has *successfully moved* before time t. Depth is initially 0. Entering at s is considered a movement, so robots entering s have depth 1.

$d(A_i^{\mathcal{P}(n)}, t)$ is similarly defined with respect to $\mathcal{P}(n)$.

Definition 6.3.4. Let T be a tree graph environment (such as $\mathcal{P}(n)$) with source vertex s. A vertex v of T becomes **slow** at time t if a mobile robot on v was activated and found no vertex it could move to, and also, either v is a leaf of T or all of its descendants in T are slow at time t.

A robot A_i is **slow** at time t if it is located at a slow vertex at time t.

Definition 6.3.5. The **slow makespan** of S on T, M^T_{slow}, is the first time all vertices of T are slow when simulating the event order S.

G is not always a tree, but given a fixed event order S, we can associate to S a spanning tree of G, T_S, containing $G(t)$ as a subtree for all times t. Lemma 6.3.1 says robots only use edges of T_S, so we may define the slow makespan of S on the G-simulation as the slow makespan on T_S. Slow makespan is clearly also defined for the $\mathcal{P}(n)$-simulation. Furthermore, M^G_{slow} is an upper bound on the (regular) makespan of the G-simulation, since every vertex must have a settled robot before it becomes slow and, as the settled robots of G never move, they cannot be deleted by S.

Our motivation for introducing slow makespan is that we wish to show $\mathcal{P}(n)$ is the environment that maximizes slow makespan on n vertices. However, it does not maximize normal makespan (see Table 6.2 for an example).

Lemma 6.3.6. *A slow robot A^G_i is forever unable to move and never deleted in the event order S.*

Proof Only robots attempting to move can be deleted. If A^G_i is at a leaf of T_S, it can never move, since its parent vertex in T_S contains a settled robot marking the vertex of a robot in a different location, and settled robots are never deleted. Hence, A^G_i is never deleted. Slow vertices propagate upwards from the leaves of T_S, so the statement of the lemma follows by induction. ■

Proposition 6.3.7. *For any event order S, $M^G_{slow} \leq M^{\mathcal{P}(n)}_{slow}$.*

An intuitive argument for this proposition is that if the spanning tree T_S of G is not $\mathcal{P}(n)$, then some vertex v of T_S must have multiple descendants, hence robots entering v will be able to branch to different neighbours and v is less likely to be blocked. Consequently, robots will enter G faster than $\mathcal{P}(n)$, and so $M^G_{slow} \leq M^{\mathcal{P}(n)}_{slow}$. We need to formalize this intuition into an argument that holds for any event order S. It turns out there are many subtleties involving asynchronicity, settling and crashing which make this not straightforward, and we require a rather technical argument. (Such subtleties are also why it is simpler to compare the environments $G, \mathcal{P}(n), \mathcal{P}(\infty), \mathcal{P}^*(\infty), B$ rather than compare G to B directly.)

We prove Proposition 6.3.7 by induction on the *meaningful event times* $t_0, t_1, \ldots$ in the event order S. We show the following statements to be true for non-deleted robots at all times t_m:

(a) If A^G_i is not slow or settled, then $d(A^G_i, t_m) \geq d(A^{\mathcal{P}(n)}_i, t_m)$.

(b) If $A^{\mathcal{P}(n)}_i$ is slow or settled, then $A^G_i(t_m)$ is slow or settled, and $d(A^G_i, t_m) \leq d(A^{\mathcal{P}(n)}_i, t_m)$.

We note that both statements are (trivially) true at time t_0, as no event has occurred yet.

Lemma 6.3.8. *If statement (b) is true up to time t_m, settled and slow robots of $\mathcal{P}(n)$ neither move nor get deleted as a result of an event of S scheduled for time t_m (i.e., the robots still exist and are in the same place at time t_{m+1}).*

Assuming (a) and (b) hold at all times, let us see how to infer Proposition 6.3.7. If a vertex becomes slow at some time t, it must contain a settled and a mobile robot, both of whom become slow. Lemma 6.3.8 says that slow and settled robots of $\mathcal{P}(n)$ never get deleted. Hence, the first time there are $2n$ slow robots on $\mathcal{P}(n)$ (two at every vertex) is $M_{slow}^{\mathcal{P}(n)}$. Statement (b) implies that if $\mathcal{P}(n)$ has $2n$ slow robots, G must also contain $2n$ slow or settled robots. It is immediate to verify that this can only happen when G has $2n$ slow robots. Hence, at time $M_{slow}^{\mathcal{P}(n)}$, G has $2n$ slow robots–two at every vertex. The inequality $M_{slow}^{\mathcal{P}(n)} \geq M_{slow}^{G}$ follows by definition. ■

Lemma 6.3.9. *If statements (a) and (b) are true up to time t_m, statement (a) is true at time t_{m+1}.*

Lemma 6.3.10. *If statements (a) and (b) are true up to time t_m, statement (b) is true at time t_{m+1}.*

Proofs of the lemmas can be found in the "Analysis details" section (Section 6.5).

$\mathcal{P}(n)$ versus $\mathcal{P}(\infty)$

We wish to bound $M_{slow}^{\mathcal{P}(n)}$ (which is determined by the event order S). We do this by comparing simulations of S on different environments. To start, let $\mathcal{P}(\infty)$ be the path graph with infinite vertices, and where $s = v_1$. We may simulate S on $\mathcal{P}(\infty)$ as we did on $\mathcal{P}(n)$.

Lemma 6.3.11. *For any event order S simulated on $\mathcal{P}(n)$ and $\mathcal{P}(\infty)$ and any time $t < M_{slow}^{\mathcal{P}(n)}$, $\mathcal{P}(n)$ and $\mathcal{P}(\infty)$ contain the exact same number of robots.*

Proof The configuration of robots in the first n vertices of $\mathcal{P}(n)$ and $\mathcal{P}(\infty)$ is identical until v_n becomes slow in $\mathcal{P}(n)$. After v_n becomes slow, the configuration of robots in the first $n-1$ vertices is still the same in both graphs until a robot in v_{n-1} is prevented from moving by a robot in v_n, meaning v_{n-1} becomes slow. By induction, the configuration of robots in the first k vertices of both graphs is identical until v_k in $\mathcal{P}(n)$ becomes slow (we use Lemma 6.3.8 to infer that the slow robots at v_{k+1} are never deleted). Hence, until v_1 becomes slow, robots enter at the same times in $\mathcal{P}(n)$ and $\mathcal{P}(\infty)$. v_1 becomes slow precisely at time $M_{slow}^{\mathcal{P}(n)}$. ■

$\mathcal{P}(\infty)$ versus $\mathcal{P}^*(\infty)$

We simulate S on the environment $\mathcal{P}^*(\infty)$. $\mathcal{P}^*(\infty)$ is $\mathcal{P}(\infty)$ with the modification that there is at time $t = 0$ a settled robot at every vertex v_i. The settled robot at v_i marks v_{i-1}. These "dummy" robots are never activated, and are not of the indexed robots $A_1, A_2, \ldots$. Because there is already a settled robot at every vertex, the robots $A_1, A_2, \ldots$ never become settled. Call this environment $\mathcal{P}^*(\infty)$. Lemma 6.3.12 shows $\mathcal{P}^*(\infty)$ is strictly slower than $\mathcal{P}(\infty)$:

Lemma 6.3.12. *For any event order S and at any time t, the amount of **mobile**-state robots in $\mathcal{P}^*(\infty)$ at time t is at most the total amount of robots in $\mathcal{P}(\infty)$.*

$\mathcal{P}^*(\infty)$ versus totally asymmetric simple exclusion

We bound the arrival rate of robots at $\mathcal{P}^*(\infty)$ by another, even slower process. This process, B, takes place on the path graph $\mathcal{P}(\infty)$ where we also have non-positive vertices $v_0, v_{-1}, v_{-2}, \ldots$, and such that there is an edge (v_i, v_{i+1}) for every i. Like $\mathcal{P}^*(\infty)$ there is initially a settled robot at every vertex, marking the vertex before it. Unlike the other processes, robots do not enter at s: the robot A_i begins inside the graph environment as a mobile robot located at v_{-i+1}. To compare B with $\mathcal{P}^*(\infty)$, we count the robots that cross the edge (v_0, v_1). There is one more crucial feature of B: robots are never deleted from B. Scheduled robot deletions at S are treated as a regular activation of the robot. Besides these differences, S can be simulated on B as before.

Lemma 6.3.13. *For any event order S and at any time t, the number of mobile robots that crossed the (v_0, v_1) edge of B is at most the number of robots that entered or were deleted before entering $\mathcal{P}^*(\infty)$.*

Recall that S is an event order of some execution of Algorithm 6.1 on the graph environment of interest, G. We may randomly sample S by running Algorithm 6.1 on G and logging the events.

The stochastic process resulting from simulating a *randomly sampled* event order S on B is called a *totally asymmetric simple exclusion process* (TASEP) with step initial condition, first introduced in [Spi91]. In this process, robots (called also "particles") are activated at exponential rate 1 and attempt to move rightward whenever no other robot blocks their path. This is precisely the outcome of simulating S on B (since robot activations that lead to a deletion in the other processes are treated as a regular activation in B).

In TASEP with step initial condition, let us write B_t to denote the number of robots that have crossed (v_0, v_1) at time t. It is shown in [Ros81] that B_t converges to $\frac{1}{4}t$ asymptotically almost surely (i.e., with probability 1 as $t \to \infty$). [Joh00] shows that the deviations are of order $t^{1/3}$. Specifically we have in the limit:

$$\lim_{t\to\infty} \mathbb{P}(B_t - \frac{t}{4} \leq 2^{-4/3} s t^{1/3}) = 1 - F_2(-s) \tag{6.1}$$

Valid for all $s \in \mathbb{R}$, where F_2 is the Tracy-Widom distribution and obeys the asymptotics $F_2(-s) = O(e^{-c_1 s^3})$ and $1 - F_2(s) = O(e^{-c_2 s^{3/2}})$ as $s \to \infty$. We employ Equation 6.1 and the prior analysis to prove Theorem 6.1:

Proof Let G be a graph environment with n vertices. Let S be the randomly sampled event order of an execution of Algorithm 6.1 on G. We will bound the slow makespan, M^G_{slow}.

We simulate S over the environments $\mathcal{P}(n)$, $\mathcal{P}(\infty)$, $\mathcal{P}^*(\infty)$, and B. From Lemma 6.3.13 we know that at all times the number of robots that crossed the (v_0, v_1) edge of B, meaning B_t, is less than the number of robots that entered $\mathcal{P}^*(\infty)$ or were deleted before entering. At most $ct/4$ robots are deleted by time t, so the number of mobile robots at $\mathcal{P}^*(\infty)$ at time t is at least $B_t - ct/4$. Lemmas 6.3.11 and 6.3.12 imply this is at least the number of robots at $\mathcal{P}(n)$ at any time $t < M^{\mathcal{P}(n)}_{slow}$.

At any time t, there cannot be more than $2n$ robots at $\mathcal{P}(n)$. Hence, if $B_t - ct/4 > 2n$, then $t \geq M^{\mathcal{P}(n)}_{slow}$. By Proposition 6.3.7, we shall then also have $t \geq M^G_{slow}$.

Write $t_n = 8((1-c)^{-1} + n^{-1/3})n$. We wish to show t_n is an upper bound on M^G_{slow} asymptotically almost surely, which is precisely the statement of Theorem 6.1. To show this, we are interested in $X_n = \mathbb{P}(B_{t_n} \leq 2n)$, the probability that B_{t_n} is less than $2n$ at time t_n. Showing X_n tends to 0 as $n \to \infty$ completes our proof. Define the probability

$$p(n, s) = \mathbb{P}(B_{t_n} - \frac{t_n}{4} \leq 2^{-4/3} s t_n^{1/3}) \tag{6.2}$$

$p(n, s)$ is the parametrized left innermost part of Equation 6.1 with $t = t_n$ (n is a positive integer). Note that $p(n, s)$ is monotonic increasing in s. Define $s_n = (2n - t_n/4)2^{4/3} t_n^{-1/3}$. By algebra, we have $X_n = p(n, s_n)$. Fix any constant $\mathcal{S}^*$ and define $Y_n = p(n, \mathcal{S}^*)$. Again by algebra, s_n tends to $-\infty$ as $n \to \infty$. Hence, for a large n, we must have $s_n < \mathcal{S}^*$ and therefore $X_n \leq Y_n$ (by the monotonicity of $p(n, s)$). By Equation 6.1, Y_n tends to $1 - F_2(-\mathcal{S}^*)$ as $n \to \infty$. Hence X_n is at most $1 - F_2(-\mathcal{S}^*)$ in the limit. By taking $\mathcal{S}^* \to -\infty$ we see that X_n in the limit is at most $1 - F_2(\infty) = 0$.■

We note that slow makespan can be nearly equal to makespan (see Table 6.2, or consider a path graph $\mathcal{P}(n)$ the source vertex placed at $s = v_2$ and robots initially moving rightwards). Hence, one does not "miss out" on much by using it to bound makespan.

6.3.2 Synchronous time and multiple sources

We describe extensions of our results to two settings.

Synchronous time. We may consider a synchronous time setting that is discretized to steps $t = 1, 2, \ldots$ such that at every step, all the robots activate at once. In this setting, Algorithm 6.1 ends up exploring just one branch of the tree at a time, like depth-first-search; so no two robots ever attempt to enter the same vertex. Analysis similar to the asynchronous case shows that robots then enter at rate $t/2$ (instead of

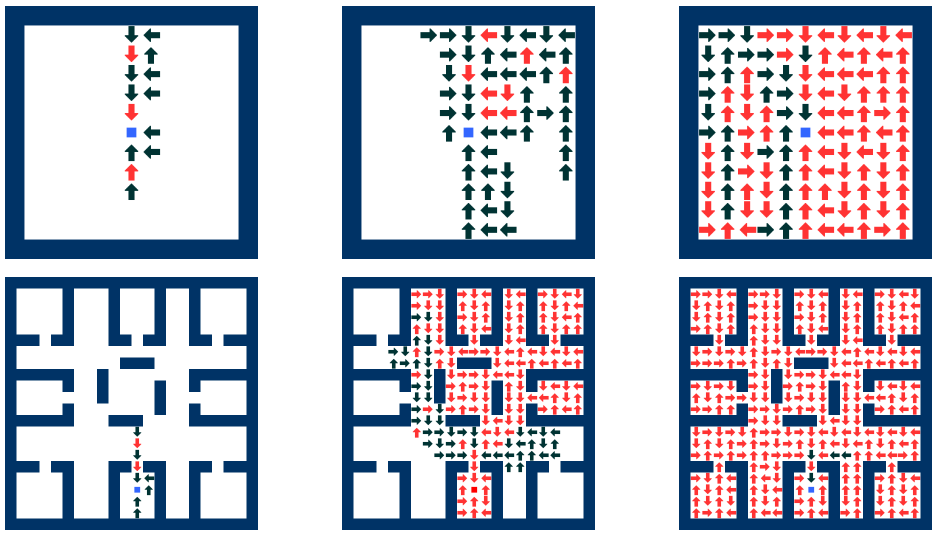
approximately $t/4$) on $\mathcal{P}(n)$, and analogous reasoning to Lemma 6.3.7 and Theorem 6.1 gives an upper bound of $4(1-c)^{-1}n$ on the makespan of a graph with n vertices, assuming $ct/2$ adversarial events. Consider the path graph $\mathcal{P}(n)$ with $s = v_2$ (not the usual $s = v_1$), and where the robots first fill the vertices $v_3, v_4, \ldots$ with a two- layer before reaching v_1. The synchronous makespan of this environment is asymptotically $4(1-c)^{-1}n$. Hence, the bound on the makespan in the synchronous case is exact.

Multiple source vertices. Instead of just having a single source vertex s, we may consider environments with multiple source vertices such that each of them corresponds to its own set of robots $A_1, A_2, \ldots$ entering over time. In asynchronous time, Lemma 6.3.1 can be generalized to show that $G(t)$ is then a *forest*, and the robots attempt to create a spanning forest of G. The technique in this paper can be generalized to show that the makespan bound of Theorem 6.1 holds. In general graph environments multiple sources may not improve the makespan by much. For example, consider the path graph $\mathcal{P}(n)$ with k sources on $v_1, v_2, \ldots v_k$. The makespan of this graph is bounded below by the makespan of the path graph $\mathcal{P}(n-k-1)$ with a single source vertex v_1.

6.4 Simulation and evaluation

For empirical confirmation of our analysis, we numerically simulated our algorithm on a number of environments. On these environments, we measured the makespan and the percentage of robots that crashed for the parameters $c = 0, 0.25, 0.75$, averaging them over 30 simulations per configuration and rounding to the nearest integer. Data on several environments is found in Table 6.2. Figure 6.3 shows stills from some simulations.

Figure 6.3 An execution of Algorithm 6.1 on (a) an 11×11 square grid and (b) an "indoor" environment. The legend is same as Figure 6.1.



From the data, it is clear that makespan is affected by the shape of the environment

	n	$c = 0$	$c = 0.25$	$c = 0.75$
Figure 6.1	62	272;373	321;446 (18%)	555;743 (52%)
Figure 6.3, (a)	121	463;554	484;586 (13%)	715;921 (39%)
Figure 6.3, (b)	300	1791;1907	2154;2281 (19%)	3894;4116 (56%)
$\mathcal{P}(300)$	300	1677;2325	1940;2883 (23%)	3727;6147 (66%)

Table 6.2: The makespan and slow makespan of Algorithm 6.1 over the environments in the referred-to Figures and over the path graph of length 300, $\mathcal{P}(300)$. We vary the crash density parameter c. The cell format is *makespan ; slow makespan (% of robots crashed)*. The column "n" gives the number of vertices in the environment.

and by c. We see that an increase in the percentage of robots crashed scales makespan up gracefully, and that spacious environments generally have lower makespans. We also confirm that the slow makespans are always lower than the bound of Theorem 6.1. Closest to the bound is the scenario where the environment is the path graph $\mathcal{P}(300)$ and $c = 0$, in which case slow makespan is almost exactly the bound, $8 \cdot 300$. This is consistent with our analysis that the $\mathcal{P}(n)$ environment has the largest slow makespan. It also verifies that Theorem 6.1 gives a correct upper bound. Such data further suggests that for spacious environments, and for large c, performance on average is better than the *worst-case* performance guarantee of Theorem 6.1. In the simulations, we did not choose our adversarial events to be maximally obstructing, but rather crashed robots arbitrarily–a cleverer adversary would cause the makespan and slow makespan to be closer to the worst-case (and cause a larger percentage of robots to crash).

6.5 Analysis details

Reminder:

(a) If A_i^G is not slow or settled at time t_m, then $d(A_i^G, t_m) \geq d(A_i^{\mathcal{P}(n)}, t_m)$.

(b) If $A_i^{\mathcal{P}(n)}$ is slow or settled at time t_m, then $A_i^G(t_m)$ is slow or settled, and $d(A_i^G, t_m) \leq d(A_i^{\mathcal{P}(n)}, t_m)$.

6.5.1 Proof of Lemma 6.3.8

Proof Referring to the Lemma's statement, we remind that here "time t_m" refers to the configuration of agents at time t_m *before* any scheduled events. Hence, even if something is true at time t_m, we still need to show that it remains true after the events that happen at time t_m.

Let $A_i^{\mathcal{P}(n)}$ be slow or settled at time t_m. To show $A_i^{\mathcal{P}(n)}$ will not be deleted, it suffices to show the event order S will not delete A_i^G. (b) implies A_i^G is settled or slow at time t_m. Lemma 6.3.6 says S never deletes slow agents. S never deletes settled agents of G as, in our model, agents are only deleted when they move, and S obeys the rules of the model when simulated on G. Hence, S will not delete A_i^G.

Next we show that $A_i^{\mathcal{P}(n)}$ will not move as a result of an event scheduled for time t_m. If $A_i^{\mathcal{P}(n)}$ is settled, this is true by definition. Otherwise, $A_i^{\mathcal{P}(n)}$ is slow. By assumption, (b) is true at *all* times up to t_m. Hence, by the same reasoning as the above paragraph, agents of $\mathcal{P}(n)$ that became slow or settled at or prior to time t_m have not been deleted. Consequently, the argument of Lemma 6.3.6 applies also here, allowing us to conclude that agents cannot move after they become slow. In particular this applies to $A_i^{\mathcal{P}(n)}$.■

6.5.2 Proof of Lemma 6.3.9

Proof Only one event occurs at time t_m. This event is either an uninterrupted activation of an agent (meaning the agent is not deleted), or an activation that leads to a deletion. If the event is a deletion, (a) holds at time t_{m+1} trivially, so we assume that it is an uninterrupted activation.

Let A_i^G and $A_i^{\mathcal{P}(n)}$ be the agents that are activated at time t_m. The depth of any other agent is unchanged, so we need only verify (a) for these two agents. Assuming (a) it true at time t_m, it is only possible for (a) to become false at time t_{m+1} if A_i^G did not move, but $A_i^{\mathcal{P}(n)}$ did. We assume this is the case.

If A_i^G does not move as a result of its activation at time t_m, then either it is settled, in which case (a) is true and we are done, or there is a mobile agent at every neighbouring vertex in $G(t_m)$. If A_i^G is mobile and all of its neighbours are slow at time t_m, then A_i^G becomes slow at time t_{m+1} and (a) is true. Otherwise there is a mobile agent, A_j^G, that is preventing A_i^G from moving and is not slow. We must have that

$$d(A_i^G, t_{m+1}) + 1 = d(A_j^G, t_{m+1}) \tag{6.3}$$

Because A_i^G and A_j^G are always moving down a spanning tree T_S of G, hence the depth of A_j^G must be precisely one greater than A_i^G's in order to prevent movement.

Because A_j^G is not activated at time t_m, (a) and (b) are still true for it at time t_{m+1}. Because A_j^G is not slow or settled, (a) implies that

$$d(A_j^G, t_{m+1}) \geq d(A_j^{\mathcal{P}(n)}, t_{m+1}) \tag{6.4}$$

And the contrapositive of (b) implies that $A_j^{\mathcal{P}(n)}$ is not settled. However, consider the structure of the graph $\mathcal{P}(n)$: if $A_j^{\mathcal{P}(n)}$ is mobile, then since it entered before $A_i^{\mathcal{P}(n)}$, it must be further ahead. In particular, we must have

$$d(A_j^{\mathcal{P}(n)}, t_{m+1}) \geq d(A_i^{\mathcal{P}(n)}, t_{m+1}) + 1 \tag{6.5}$$

As otherwise $A_j^{\mathcal{P}(n)}$ would have prevented $A_i^{\mathcal{P}(n)}$ from moving when activated at time t_m.

(In)equalities 6.3, 6.4 and 6.5 imply $d(A_i^G, t_{m+1}) \geq d(A_i^{\mathcal{P}(n)}, t_{m+1})$. This shows (a) is true at time t_{m+1}. ■

6.5.3 Proof of Lemma 6.3.10

Proof As in Lemma 6.3.9, we can assume that the event at time t_m is the uninterrupted activation of a pair of agents A_i^G and $A_i^{\mathcal{P}(n)}$, and we need only verify that (b) is still true for this pair of agents. We separate our proof into cases.

Case 1: Assume $A_i^{\mathcal{P}(n)}$ is settled at time t_{m+1}. Because $\mathcal{P}(n)$ is a path graph and using Lemma 6.3.8, $A_i^{\mathcal{P}(n)}$ can only be settled if every non-deleted agent that entered before it is settled behind it. At time t_{m+1} (b) is still true for all agents other than A_i^G and $A_i^{\mathcal{P}(n)}$. Hence, it follows from (b) that for any non-deleted agent A_j^G where $j < i$ we have:

$$d(A_j^G, t_{m+1}) \leq d(A_j^{\mathcal{P}(n)}, t_{m+1}) \tag{6.6}$$

Algorithm 6.1 guarantees that any agent in G always neighbours a settled agent or is at the same location as a settled agent. Thus, we know that $d(A_i^G, t_{m+1}) \leq d(A_j^G, t_{m+1}) + 1$ for some settled agent A_j. Furthermore, this inequality must hold for some A_j^G that entered *before* A_i^G (i.e., $j < i$), because any settled agent that entered after A_i^G must have gone down a different branch of T_S, otherwise it would be blocked by A_i^G and unable to settle. Let $j_{max} = max_{j<i} d(A_j^G, t_{m+1})$. Then $d(A_i^G, t_{m+1}) \leq j_{max}+1$. If this is an equality, A_i^G is necessarily settled.

From Inequality 6.6 we infer

$$d(A_i^{\mathcal{P}(n)}, t_{m+1}) \geq max_{j<i} d(A_j^{\mathcal{P}(n)}, t_{m+1}) + 1 \geq j_{max} + 1 \geq d(A_i^G, t_{m+1}) \tag{6.7}$$

Where $d(A_i^{\mathcal{P}(n)}, t_{m+1}) \geq max_{j<i} d(A_j^{\mathcal{P}(n)}, t_{m+1}) + 1$ follows from the fact that $A_i^{\mathcal{P}(n)}$ is ahead of all non-deleted agents that came before it. In the case of equality, A_i^G must be settled. If A_i^G isn't settled, then the inequality is strict. Consequently, it follows from the fact that (a) holds at time t_{m+1} (Lemma 6.3.9) that A_i^G must be slow. Otherwise, (a) implies A_i^G's depth is greater than $A_i^{\mathcal{P}(n)}$'s, contradicting the inequality. Either way, (b) is true.

Case 2: Assume $A_i^{\mathcal{P}(n)}$ is slow and not settled at time t_{m+1}. If $A_i^{\mathcal{P}(n)}$ is slow at t_m, then it follows from (b) that A_i^G is slow or settled at t_m, and so activation cannot affect either of these agents, meaning (b) remains true at t_{m+1} and we are done. Thus, we may assume $A_i^{\mathcal{P}(n)}$ is not slow at time t_m.

Using Lemma 6.3.8, $A_i^{\mathcal{P}(n)}$ can only become slow at time t_{m+1} if all vertices behind it contain settled agents, and all vertices ahead of it contain two slow agents (one settled and one mobile). If $d(A_i^{\mathcal{P}(n)}, t_m)$ is k there are $n + (n - k) = 2n - k$ slow or settled agents in $\mathcal{P}(n)$ at time t_m. These $2n - k$ agents must have entered $\mathcal{P}(n)$ before $A_i^{\mathcal{P}(n)}$, because any agent that enters after $A_i^{\mathcal{P}(n)}$ must pass it to become slow or settled, and this is impossible because $A_i^{\mathcal{P}(n)}$ is not settled.

Using (b) we learn from the above that in G, at time t_m there are at least $2n - k$

settled or slow agents that entered before A_i^G. Of these, at least $n-k$ agents are slow and mobile, and have greater depth than A_i^G or are in a different branch of T_S (because they arrived before A_i^G and A_i^G could not have passed them). There are thus at most $n-(n-k)=k$ vertices A_i^G could have visited since entering G, meaning its depth is at most k, and we have $d(A_i^G, t_m) \leq d(A_i^{\mathcal{P}(n)}, t_m)$.

If this inequality is strict, then from statement (a) we learn that A_i^G is settled or slow, so (b) is true and we are done. Otherwise, $d(A_i^G, t_m) = k$. We saw there are (at least) $n-k$ slow mobile agents in G that have greater depth than A_i^G or are in a different branch of T_S. From this, we infer that any descendant of A_i^G must contain a slow mobile agent, or that A_i^G is at a leaf of T_S and has no descendants. Thus, if A_i^G is not already settled or slow, it will become slow after the activation at time t_m, since its slow descendants will prevent it from moving. This completes the proof. ■

6.5.4 Proof of Lemma 6.3.12

Proof Let A_i^* be the copy of A_i simulated over $\mathcal{P}^*(\infty)$. Let $t_0, t_1, t_2, \ldots$ be the meaningful event times of S. We show by induction that at any time t_m, for all non-deleted agents:

$$\text{either } A_i^{\mathcal{P}(\infty)} \text{ is settled or } d(A_i^*, t_m) \leq d(A_i^{\mathcal{P}(\infty)}, t_m) \tag{6.8}$$

This implies any agent that enters $\mathcal{P}^*(\infty)$ must have already or concurrently entered $\mathcal{P}(\infty)$, completing the proof.

The induction statement is trivially true at time t_0, as no event has occurred yet. We assume it is true up to time t_m, and show it remains true at t_{m+1}.

If the event scheduled for time t_m was a deletion of some agent, the statement remains trivially true (as both simulated versions of the agent are deleted). Otherwise, the scheduled event is the uninterrupted activation of some pair of agents A_i^* and $A_i^{\mathcal{P}(n)}$.

Any agent A_j where $j \neq i$ does not move, so we need only verify the inductive statement remains true for A_i^* and $A_i^{\mathcal{P}(n)}$. The only situation in which Inequality 6.8 is falsified at time t_{m+1} if it is true at time t_m is if $d(A_i^*, t_m) = d(A_i^{\mathcal{P}(\infty)}, t_m)$ and $A_i^{\mathcal{P}(\infty)}$ is mobile at time t_{m+1}, but A_i^* manages to move whereas $A_i^{\mathcal{P}(\infty)}$ is blocked by a mobile agent $A_j^{\mathcal{P}(\infty)}$. By the inductive hypothesis, $d(A_j^*, t_m) \leq d(A_j^{\mathcal{P}(\infty)}, t_m)$. Because $\mathcal{P}^*(\infty)$ is a path graph and $j < i$, we know that $d(A_j^*, t) > d(A_i^*, t)$ at all times t after A_j^* entered the environment. Hence, if $d(A_i^*, t_m) = d(A_i^{\mathcal{P}(\infty)}, t_m)$ and $A_j^{\mathcal{P}(\infty)}$ blocks $A_i^{\mathcal{P}(\infty)}$, then A_j^* must also block A_i^* when it attempts to move. This shows that the inductive hypothesis is correct at time t_{m+1}. ■

6.5.5 Proof of Lemma 6.3.13

Proof Unlike Lemma 6.3.12, here we count the number of agents that enter $\mathcal{P}^*(\infty)$, and not the number of currently existing agents that entered it. This means we count

also agents that entered at $\mathcal{P}^*(\infty)$ but were deleted. This difference is necessary for the comparison, because agents cannot be deleted from B.

Despite this difference, the proof is very similar to Lemma 6.3.12. One shows by induction on the meaningful event times $t_0, t_1, t_2, \ldots$ that at any time t_m, for any i such that $A_i^{\mathcal{P}^*(\infty)}$ was not deleted we have:

$$d(A_i^B, t_m) - i + 1 \leq d(A_i^{\mathcal{P}^*(\infty)}, t_m) \tag{6.9}$$

Note that $d(A_i^B, t_m) - i + 1$ is the index of the vertex of A_i^B at time t_m. If A_i^B crossed (v_0, v_1) we must have $d(A_i^B, t_m) - i + 1 \geq 1$. Recalling that if $A_i^{\mathcal{P}^*(\infty)}$ is outside of G at time t then $d(A_i^{\mathcal{P}^*(\infty)}, t) = 0$, we see by (6.9) that crossing (v_0, v_1) can only happen if $A_i^{\mathcal{P}^*(\infty)}$ entered $\mathcal{P}^*(\infty)$, or if $A_i^{\mathcal{P}^*(\infty)}$ was deleted before entering. Hence, the Lemma follows from (6.9).

Let us show (6.9) holds by induction. It holds trivially for all i at t_0. Now, assume (6.9) holds at time t_m, and we will show it holds at time t_{m+1}.

Suppose the pair of agents activated at t_m is $A_i^{\mathcal{P}^*(\infty)}$ and A_i^B. Then these are the only agents for which (6.9) might be false at t_{m+1}. Assuming (6.9) is true at t_m, it can only become false at t_{m+1} if $d(A_i^B, t_m) - i + 1 = d(A_i^{\mathcal{P}^*(\infty)}, t_m)$, but A_i^B successfully moves as a result of activation at time t_m whereas $A_i^{\mathcal{P}^*(\infty)}$ does not and also is not deleted. If $A_i^{\mathcal{P}^*(\infty)}$ does not move this means some $A_j^{\mathcal{P}^*(\infty)}$, $j < i$ is blocking it. Hence, we must have $d(A_j^{\mathcal{P}^*(\infty)}, t_{m+1}) = d(A_i^{\mathcal{P}^*(\infty)}, t_{m+1}) + 1$. By the inductive hypothesis we have $d(A_j^B, t_{m+1}) - j + 1 \leq d(A_j^{\mathcal{P}^*(\infty)}, t_{m+1})$. Since $j < i$, A_j^B is always ahead of A_i^B, meaning $d(A_i^B, t_{m+1}) - i + 1 < d(A_j^B, t_{m+1}) - j + 1$. Combining these (in)equalities we get $d(A_i^B, t_{m+1}) - i + 1 < d(A_i^{\mathcal{P}^*(\infty)}, t_{m+1}) + 1$, hence $d(A_i^B, t_{m+1}) - i + 1 \leq d(A_i^{\mathcal{P}^*(\infty)}, t_{m+1})$. This completes the proof by induction of (6.9). ■

6.6 Discussion

In swarm robotics, where one must coordinate an enormous robotic fleet, we must anticipate many faults, such as crashing and traffic jams. Because robots in the swarm are usually assumed to be autonomous and have limited computational power, complex techniques for handling such faults are not necessarily feasible. Hence, it is important to ask whether simple rules of behaviour can be effective. To this end, we investigated the problem of covering an unknown graph environment, and constructing an implicit spanning tree, with a swarm of frequently crashing robots. We showed a simple and local rule of behaviour that enables the swarm to quickly and reliably finish this task in the presence of crashes. The swarm's performance degrades gracefully as crash density increases.

We outline here several directions for future research. First, our model interprets the "swarm" part of swarm robotics as a vast and redundant fleet of robots that can be dispersed into the environment over time. We used this model for uniform dispersal, but

it would be interesting to adapt it to other kinds of missions, and design algorithms for those missions that can handle crashes or other forms of interference. For example, in Chapter 3, mobile agents entering at a source node s over time sequentially pursue each other to discover shortest paths between s and some target node t. The "algorithm" in Chapter 3 succeeds even if some of the agents are interrupted and have their location changed.

Next, in this chapter, we made the simplifying assumption that the environment of the robots is discrete. If the robots instead attempted to cover a continuous planar domain by an algorithm similar to ours, the robots would need to construct a shared discrete graph representation of the environment through the settled robots in $G(t)$ and their markings. We believe that our algorithm can readily be extended to such settings.

Lastly, can we exploit the large number of robots in a swarm to handle other kinds of errors? There are many situations and modes of failure that can be discussed, such as Byzantine robotic agents, or dynamic changes to the environment.

Chapter 7

Swarm Robotics III: Physical Sorting

Chapters 5 and 6 explored the topic of swarm uniform dispersion. Here we explore a different problem, inspired by the idea of efficiently routing self-driving vehicles on a freeway to their destinations, which we term "physical sorting". Given a collection of red and blue mobile agents located on two grid rows, we seek to move all the blue agents to the far left side and all the red agents to the far right side, thus *physically sorting* them according to color. The agents all start on the bottom row. They move simultaneously at discrete time steps and must not collide. Our goal is to design a centralized algorithm that controls the agents so as to sort them in the least number of time steps. We derive an *exact* lower bound on the amount of time any algorithm requires to physically sort a given initial configuration of agents, and present a centralized decisionmaking algorithm that matches this lower bound.

This chapter is based on the work [RAB21]. We wish to emphasize that centralized algorithms are significantly more powerful than the ant-like algorithms we stuck to throughout this work, and explicitly do not belong in the ants paradigm. What relates the work reported here to the themes of this dissertation is our showing that this powerful, centralized algorithm performs only slightly better than a distributed, ant-like physical sorting algorithm. Specifically, we show that whenever the initial agent configuration is "normal", meaning the leftmost agent is red and the rightmost agent is blue, a very straightforward, ant-like, decentralized and local sensing-based physical sorting algorithm is at most 1 time step slower than a centralized instance-optimal algorithm. We did something similar in Chapter 5 when we placed centralized and ant-like algorithms on similar grounds by showing that even a centralized algorithm cannot optimize travel in general grid environments (Proposition 5.3.9), whereas an ant-like algorithm suffices for optimizing travel in simply connected environments.

In addition to the above, we wish to highlight two points connecting the results we obtain in this chapter with the broader ideas in this dissertation:

1. Our model of agent motion is similar to that of previous chapters and is TASEP-related. In fact, a special case of one of our results (Corollary 7.3) can be used to compute the time it takes an arbitrary TASEP particle configuration with synchronized waking times to get from one end of a row to the other, assuming particles always move right (see Section 7.1). Although this corollary is fairly straightforward, we could not find a similar result in the TASEP literature.

2. Despite the majority of our efforts being dedicated to analyzing a *centralized* sorting algorithm, our mathematical analysis of physical sorting is fundamentally agent-based and uses many techniques from previous chapters, the most notable of which is *exchangeability* (which we use to define "labels" that agents exchange between each other), suggesting that these techniques have broader applicability.

7.1 Introduction

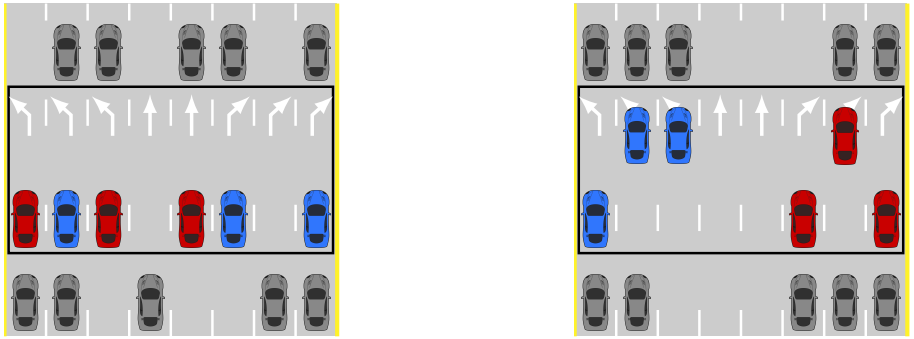
Suppose a number of mobile agents are moving on a row. Some of the agents need to travel left, and the other agents need to travel right to arrive at their destination. The agents are not allowed to collide, but have access to another adjacent, initially empty row that they can use to manoeuvre past each other. What is the most efficient way for the agents to achieve their goal and arrive at their desired left-side or right-side destinations?

In this chapter we study a discrete formalization of this problem. Given a collection of red and blue mobile agents located on parallel grid rows of equal length, we seek a centralized algorithm to move all the blue agents to the far left side and all the red agents to the far right side columns, thus *sorting* the agents according to color (see Figure 7.2). We assume all agents are initially located on the bottom row. Agents can move simultaneously at discrete time steps $T = 0, 1, \ldots$, and must not collide with each other (two agents may never occupy or attempt to move to the same location). Our goal is to design a centralized algorithm that controls the agents so as to sort them in the least number of time steps.

This problem first arose as part of ongoing research into traffic management algorithms for self-driving vehicles on a freeway. In the context of traffic management, the two rows represent a moving subsection of an upwards-facing freeway. The subsection tracks a set of vehicles all driving at the same forward velocity. We assume that freeway exits might be placed to the left or right of the road. Thus, it is necessary to shift all vehicles that need to exit the freeway leftwards or rightwards ahead of time depending on their desired exit direction, i.e., to "sort" them (see Figure 7.1). In this chapter we focus on the special case where the top row is initially empty, which we believe to be independently interesting.

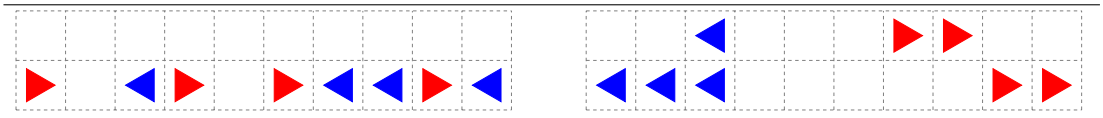
Similar problems in vehicular control, warehouse management, and combinatorial puzzles have been investigated in the literature and might broadly be referred to as

Figure 7.1 Vehicles driving on a freeway. The blue vehicles want to exit the freeway via an upcoming left exit (not illustrated), and the red vehicles want to exit the freeway via an upcoming right exit. Vehicles need to shift their position to the left or right ahead of time to prepare for exiting the freeway. (a) shows an unsorted configuration, and (b) shows the sorted configuration, after the vehicles have adjusted their positions.



(a) **(b)**

Figure 7.2 (a) illustrates an initial configuration of agents. (b) illustrates a sorted configuration (there are many possible such configurations).



(a) **(b)**

"physical sorting" problems [ZGM16; KHM+15; LV10; SCZ16; SCZ14; RW90; JS+79]. In physical sorting problems, a collection of mobile agents that occupy physical space must bypass each other without colliding in order to arrive at some predefined sorted configuration.

A trivial algorithm that accomplishes our agents' sorting task is the following: move all the agents of one color (say, red) to the top row; then let the red agents move right and the blue agents move left in their respective row (Figure 7.3, a-e). This algorithm does not require complex computation nor even centralized decision-making– it is a straightforward, decentralized, local strategy that can be executed by simple autonomous agents without requiring any global knowledge on the agent configuration.

How does the above simple algorithm fare compared to an optimal centralized sorting algorithm? A priori, since the number of possible strategies for the agents is enormous, one would expect far better sorting strategies are available. However, we shall prove the surprising result that this "trivial" distributed strategy is at most one time step short of optimal for a very large class of initial agent configurations called *normal configurations* (configurations where the leftmost agent wants to go right and the rightmost agent wants to go left), and is in fact optimal over such configurations assuming we choose the correct color to move to the spare row.

In the general case where we also consider non-normal configurations, we show that

the optimal makespan is determined by the maximal normal subconfiguration, and find a provably optimal sorting algorithm for the agents (Section 7.6). Furthermore, we derive an **exact** lower bound for the amount of time it takes to sort the agents given any starting configuration (Theorem 7.1). Our proposed optimal algorithm matches this lower bound, thus it is instance optimal in the sense of [ABC17], attaining the best possible sorting time for any initial configuration.

The algorithm (Algorithm 7.2) can be understood as blending two strategies: inside the maximal normal subconfiguration, we split the agents into rows based on their desired direction of motion, according to the aforementioned "trivial" strategy. Outside this subconfiguration, agents split between both rows to move faster regardless of their color (Figure 7.9). The idea behind this algorithm is simple to understand, but the implementation requires several delicate caveats which are discussed throughout this chapter.

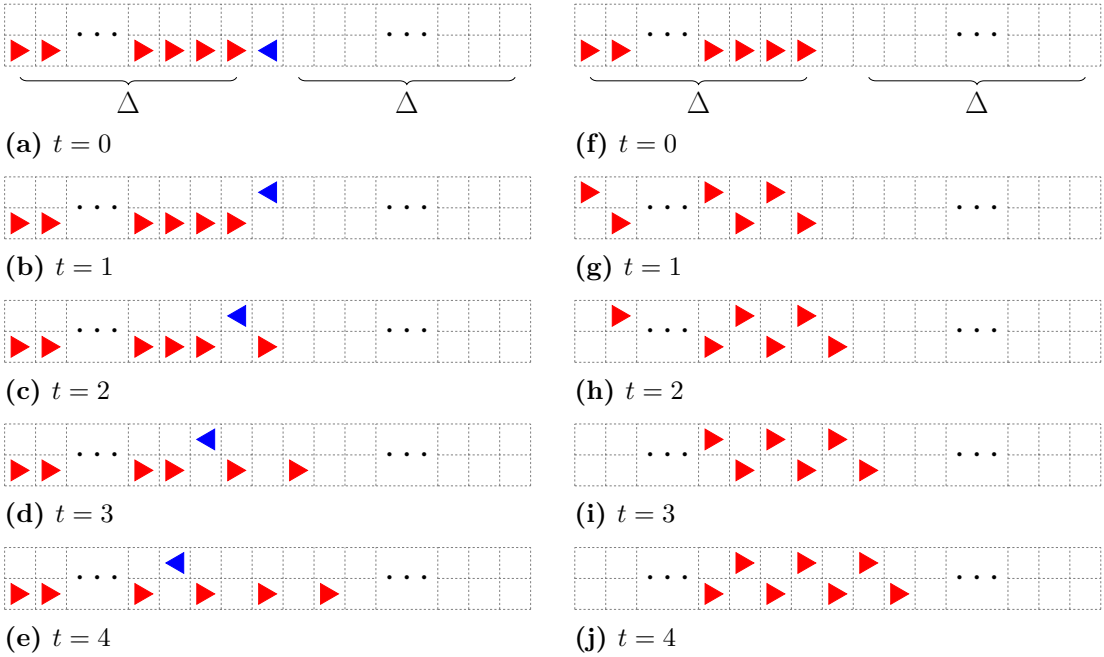
Physical sorting can sometimes be unintuitive: the addition of a single agent can completely change the optimal strategy and double the time to completion of the sorting (Figure 7.3). Furthermore, when attempting to derive lower bounds on the makespan, the infinite set of strategies available to the agents makes it difficult to keep track of each agent, which greatly complicates the mathematical proofs. We overcome these complexities by identifying a small set of *critical agents* (see Definition 7.4.3) and showing, roughly, that only the movements of the critical agents at a given time step can affect any algorithm's makespan.

We believe that the expression for the lower bound obtained in Theorem 7.1 is independently interesting. As an example application, Corollary 7.3, which is a small special case of the expression, computes the amount of time it takes for a "traffic jam" of agents on a single row to get to the right (left) side of the row, assuming each agent moves right (left) at every time step where there is no agent in front of them (see Figure 7.4). This corollary relates to the study of the *totally asymmetric simple exclusion process* in statistical mechanics, which has been used to study transport phenomena such as traffic flow and biological transport [CMZ11; KK10; CSS00]. The lower bound of Theorem 7.1 can be applied to compute the time it takes an arbitrary TASEP particle configuration with synchronized waking times to get from one end of the row to the other, assuming $p_{right} = 1$ (probability of moving right) and $p_{left} = 0$. Although this corollary is fairly straightforward, we could not find a similar result in the TASEP literature.

7.2 Related Work

Various "physical sorting" problems appear in the literature [ZGM16; KHM$^+$15; LV10; SCZ16; SCZ14] with applications to warehouse logistics, where loads must be efficiently moved to different ends of a warehouse (in particular, our assumptions resemble those of puzzle based storage systems [GK07]); servicing, where mobile robots self-sort according

Figure 7.3 (a)-(e) illustrate the first five time steps of an optimal strategy for sorting a normal initial configuration with a single blue agent placed in front of a red agent group. No sorting is possible in faster than 2Δ ticks. Subfigures (f)-(j) illustrate non-normal initial configuration that can be sorted in $\Delta + 1$ ticks via splitting the agents between the rows in an alternating fashion (Algorithm 7.2). The first five time steps of this splitting strategy are shown. Both initial configurations share a very long sequence of red agents. The only difference is a lone blue agent. However, the optimal sorting times are strikingly different.



(a) $t = 0$ **(f)** $t = 0$

(b) $t = 1$ **(g)** $t = 1$

(c) $t = 2$ **(h)** $t = 2$

(d) $t = 3$ **(i)** $t = 3$

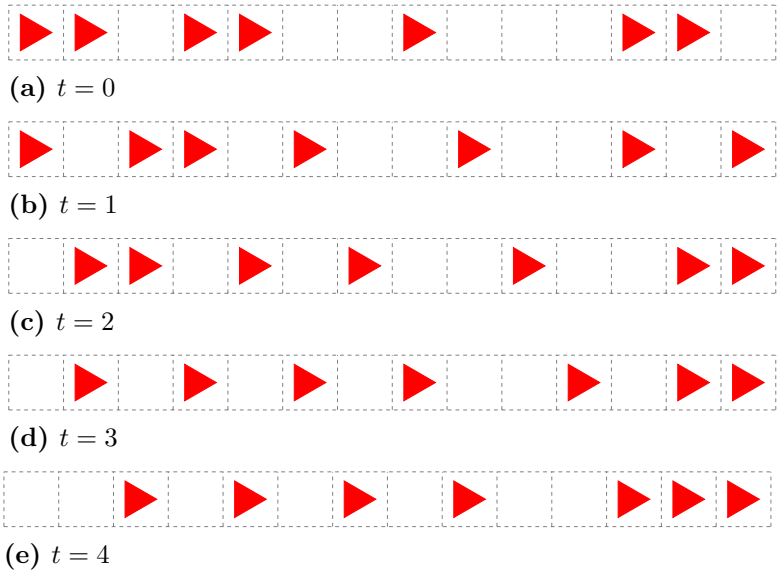
(e) $t = 4$ **(j)** $t = 4$

to some priority ordering; assembling, where tasks should be carried in a predefined order which respects physical space; and transportation. In such problems, mobile agents (robots, vehicles, etc.) that take up physical space are required to attain some kind of sorted configuration while avoiding collisions and maneuvering around each other, making their decisions in either a distributed [ZGM16] or centralized [PSS18] fashion. For example, similar to us, in [PSS18], the authors consider a task of sorting mobile vehicles in two rows, developing approximation algorithms and proving computational hardness results. Their goal, however, is to *split* the vehicles between the rows according to their color as fast as possible, as opposed to our goal of bringing robots to the left-hand or right-hand side depending on their color.

The famous "15 puzzle", where numbered tiles must be slid across a grid until they are ordered, can also be seen as a physical sorting task for which algorithms and computational hardness results are available [JS$^+$79; RW90].

Lane changing has been intensively studied in the past in many traffic models [HOU95; CT94; NGGD08; ASS16], with recent results in both centralized and distributed control protocols [VAT$^+$16; Eke19; SSW10].

Figure 7.4 Subfigures (a)-(e) illustrate a single row model with agents moving in one direction. The model could be viewed as a TASEP with $p_{\text{right}} = 1$ and $p_{\text{left}} = 0$. In the provided example, all agents reach the far-right side on the 9th time step, which is equal to what we define as the f_{max} value of the configuration (see Definition 7.4.2 and Corollary 7.3)



(a) $t = 0$

(b) $t = 1$

(c) $t = 2$

(d) $t = 3$

(e) $t = 4$

As previously mentioned, a side application of our results (Corollary 7.3) relates to *totally simple exclusion processes* (TASEPs) in statistical mechanics [CMZ11; KK10; CSS00]. TASEPs have been used, in particular, as an idealized model that captures many of the phenomena of real-world vehicular traffic. The general model we present is, however, not directly related to any TASEP model, since we study deterministic mobile agents that act according to an intelligent, centralized algorithm, rather than a predetermined stochastic process.

Broadly speaking, our mathematical modelling of the problem (Section 7.3) also relates to various discrete grid-like settings in multi-robot and multi-agent systems, wherein a large number of robots whose spatial locations are represented as coordinates on a grid-like region all move synchronously according to a local or centralized algorithm while avoiding collisions. The goal of agents in these settings can be, for example, fast deployment, gathering, or formation on an a priori unknown grid environment [AB19b; AB20; BDS08; APB18; DPV15; FYO+15; HL20].

7.3 Model

We are given n mobile agents on a $2 \times m$ grid environment, such that n_1 agents are red and n_2 agents are blue ($n_1 + n_2 = n$). Every agent begins in an (x, y) coordinate of the form $(\cdot, 1)$ (see Figure 7.2a), starting at $x = 1$. A column of the environment is called *red-occupied* if it contains at least one red agent and no blue agents, *blue-occupied* if it contains at least one blue agent and no red agents, *mixed* if it contains both colors

of agents, and *empty* otherwise. The goal of the agents is to move to a configuration such that the red-occupied columns are the rightmost columns, and the blue-occupied columns are the leftmost columns. Formally, the following conditions must be fulfilled:

1. There are no mixed columns.
2. Every blue-occupied column is to the left of every red-occupied column and every empty column.
3. Every red-occupied column is to the right of every blue-occupied column and every empty column.

When the agents achieve such a configuration we say the system is *sorted* (see Figure 7.2b), as it separates all agents in the configuration to the far left or far right columns of the environment based on color. It might be desirable to require, in addition to the above three conditions, that all agents end up on the bottom row (just as they began): in all algorithms we present here, this can be achieved at the cost of at most one additional time step that moves all the robots downwards, and analogous optimality results hold.

We define a model of agent motion based on common assumptions in synchronous and semi-synchronous models of traffic flow, biological systems, and the theory of mobile multi-robot systems [CMZ11; BDS08; APB18; DPV15; FYO$^+$15].

Time is discretized into steps $t = 0, 1, \ldots$. At every time step, agents may move to adjacent empty locations (up, down, left, or right, non-diagonally). Agents cannot move to a location that already contains an agent, nor can two agents move to the same empty location at the same time.

Note that, according to the above assumptions, a robot can only move into a location that is unoccupied, but two or more adjacent agents followed by an empty location cannot both move right (or left) in the same time step, meaning that such situations result in a traffic jam or "shock": the agents at the leftmost (rightmost) end of the line must wait several time steps for empty space to be created between them and the rest of the agents (see the first column of Figure 7.3). This is a common assumption in the literature on traffic flow, reflecting the fact that adjacent agents (even those receiving commands transmitted by a central algorithm) do not have perfectly coordinated clocks nor perfect motion detection systems and thus cannot begin moving at the exact same time and at the exact same speed without risking collisions.

The *beginning* of a time step refers to the configuration before any agent movements, and the *end* of the time step refers to the configuration after agent movements (so the configuration at the beginning of time $t+1$ is the same as at the end of time t). Unless explicitly stated otherwise, anywhere in this chapter, when we refer to the agents' configuration "at time t", we mean the configuration at the beginning of that time step.

The agents' actions are controlled by a sorting algorithm. The number of time steps it takes an algorithm to move the agents from the initial configuration to a sorted configuration is called the *makespan* of that algorithm with respect to the initial configuration (i.e., the makespan is the first t such that the configuration is sorted at the beginning of time step t).

Definition 7.3.1. A **normal** initial configuration is an initial configuration of agents such that the rightmost agent is blue and the leftmost agent is red.

Unless stated otherwise, we assume that the initial agent configuration is normal. We will specifically handle non-normal initial configurations in Section 7.6. Certain strategies, such as an alternating split between the rows that enables each agent to move horizontally faster, are effective in non-normal configurations but ineffective in normal configurations, thus the optimal strategy can be different. An example is shown in Figure 7.3, which compares a normal configuration to a non-normal configuration and shows an optimal strategy for each case. In Figure 7.3, the addition of a single blue agent completely alters the optimal strategy, and doubles the optimal makespan.

7.4 A lower bound on makespan

In this section, we will prove a lower bound on the makespan of any (centralized or distributed) sorting algorithm for normal configurations. In the next section, we will show this lower bound is tight for normal configurations: there is a centralized algorithm that matches it exactly.

The lower bound is defined as a function of the initial configuration of agents and empty columns. For every agent A, we denote A's location at time t as $(A_x(t), A_y(t))$. Let $\mathcal{A} = \mathcal{R} \cup \mathcal{B}$ be the set of all agents, where $\mathcal{R}$ is the set of red agents and $\mathcal{B}$ is the set of blue agents.

Definition 7.4.1. For any agent A, we define $front(A)$ and $back(A)$.

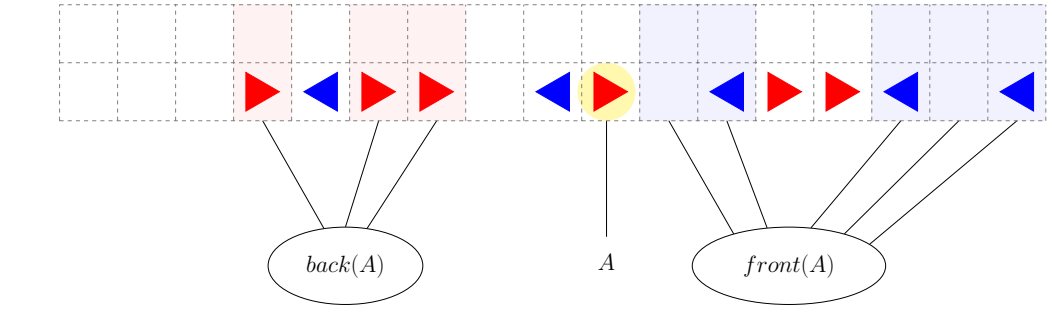
1. If $A \in \mathcal{R}$, then $front(A)$ is the set of all *not*-red-occupied columns (i.e., empty or blue-occupied) to the right right of A (i.e., with strictly greater x coordinate) and $back(A)$ is the set of all red-occupied columns to the left of A (i.e., with strictly smaller x coordinate) at time 0.

2. If $A \in \mathcal{B}$, then $front(A)$ is the set of all *not*-blue-occupied columns to the left of A and $back(A)$ is the set of all blue-occupied columns to the right of A at time 0.

item 7.4.1 is illustrated in Figure 7.5.

For any agent A we define the value $f(A)$ which is dependent on the initial configuration. We will show that every value $f(A)$ in an initial configuration is a lower bound on the makespan of *any* sorting algorithm executed over that initial configuration. Thus the maximum f_{max} of all these values is a lower bound as well.

Figure 7.5 The *front* and the *back* of the red agent A (highlighted in yellow). We see that $f(A) = 8$ (Definition 7.4.2).



Definition 7.4.2. For every agent A, we define $f(A)$ as follows (see Figure 7.5):

$$f(A) = |front(A)| + |back(A)|,$$

We further define $f_{max} = \max_{A \in \mathcal{A}} f(A)$.

Definition 7.4.3. An agent A for which $f(A) = f_{max}$ is called a **critical agent**.

We will show that in many initial configurations, f_{max} is the exact lower bound for the makespan of any sorting algorithm. However, in some situations, such as when the agents are too densely packed together and must waste time without being able to move, it is possible that the algorithm requires one extra time step to sort the agents. To derive an exact lower bound, we will have to take into account this "plus 1," whose preconditions are captured by the following definition:

Definition 7.4.4. We define $\mathcal{V}$ to equal 1 if:

1. There is both a red and a blue critical agent, or
2. There is a critical agent A such that, in the initial configuration, another agent is located immediately in front of it (i.e., at $(A_x(0)+1, 1)$ if A is red and $(A_x(0)-1, 1)$ if A is blue).

Otherwise, $\mathcal{V} = 0$.

The goal of this section and the next one is to prove the following result:

Theorem 7.1. *The makespan of any physical sorting algorithm for a given normal initial configuration is at least $f_{max} + \mathcal{V}$. This lower bound is precise; there is an algorithm that achieves sorting in exactly $f_{max} + \mathcal{V}$ time steps.*

For the sake of the proof, it is helpful to track the relative ordering of red and blue agents based on their x coordinate. To this end, we assign each red agent an integer called a *label*. At time $t = 0$, the labels depend on the relative x-coordinate position

of the agents, such that the *leftmost* red agent has the label n_1, the second-leftmost red agent has the label $n_1 - 1$, and so on, with the rightmost red agent having label 1. Two agents R and R' *exchange* their labels at time t if $R_x(t-1) \leq R'_x(t-1)$ but $R_x(t) > R'_x(t)$ (in other words, they pass each other horizontally). When this happens, R' receives the label R had at time $t-1$, and vice-versa.

We similarly define labels for blue agents, but in the *opposite* order of x coordinates, such that the *rightmost* blue agent receives label n_2, and the *leftmost* blue agent receives label 1. Label exchanging is defined as before.

Note that labels can only be exchanged by agents of the same color. When a pair of red and blue agents pass each other, labels are unchanged.

We define $ord(A, t)$ to equal the agent A's label at time t.

Let $\mathcal{P}_i^B(t)$ denote the blue agent A with $ord(A, t) = i$ at time t. When t is implicitly obvious, we will simply write $\mathcal{P}_i^B$. In the text, we will intuitively think of $\mathcal{P}_i^B$ **as having a "persistent" identity**, like an agent, and track its position. We will likewise define $P_i^R(t)$ for red agents.

Observation 7.4.5. Let $ord(A, t) = ord(A', t+1)$, where A and A' are agents of the same color. Then $|A_x(t) - A'_x(t+1)| \leq 1$, i.e. a label can not travel faster than one step horizontally in a single time tick.

Observation 7.4.5 follows from the fact that agents can only "exchange" labels by bypassing each other.

We will now prove several claims about blue agents which will establish lower bounds. Symmetrically, all such claims will hold for red agents, which will allow us to wrap up the proof at the end of the section.

We will henceforth assume the agents move according to some makespan-optimal algorithm ALG. Per Theorem 7.1, our end-goal is to show that ALG's makespan is at least $f_{max} + \mathcal{V}$.

Definition 7.4.6. A red agent R and a blue agent B are said to **meet** at time t if

$$\operatorname{sgn}(R_x(t-1) - B_x(t-1)) \in \{1, -1\}$$

and

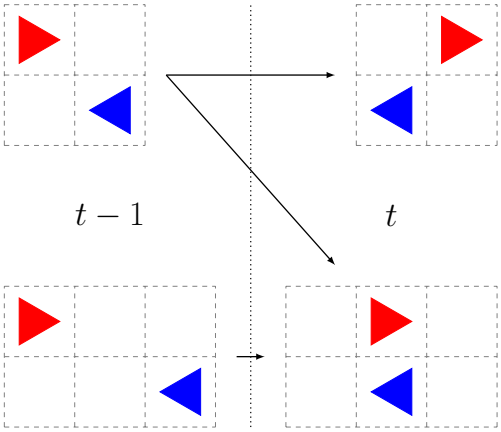
$$\operatorname{sgn}(R_x(t-1) - B_x(t-1)) \neq \operatorname{sgn}(R_x(t) - B_x(t)),$$

where $\operatorname{sgn}(\cdot)$ denotes the sign (-1, 0 or 1) of an integer. (See Figure 7.6.)

Two labels P^B and P^R are said to meet at time t if their respective agents meet at time t.

Definition 7.4.7. Define $\mathcal{V}_i^B = 0$ if during the execution of ALG, $\mathcal{P}_i^B$ decreases its x coordinate at *every* time step before it arrives at column i for the first time **and** before it meets $\mathcal{P}_{n_1}^R$. Otherwise let $\mathcal{V}_i^B = 1$.

Figure 7.6 The possible outcomes of a "meeting" between blue and red agents. On the left are two possible states of the agents at time $t-1$ before meeting, and on the right are the possible states they can transition into after meeting.



Define $\mathcal{V}_i^R = 0$ if during the execution of ALG, $\mathcal{P}_i^R$ increases its x coordinate at *every* time step before it arrives at column $n-i+1$ for the first time **and** before it meets $\mathcal{P}_{n_2}^B$. Otherwise let $\mathcal{V}_i^R = 1$.

Informally speaking, the values $\mathcal{V}_i^B$ and $\mathcal{V}_i^R$ are a measure of whether a given label ever performs a movement that will propagate backwards and slow down all the labels behind it. We will see how this occurs in Lemma 7.4.10, but the intuitive idea is this: if there is any time step where, say, the label $\mathcal{P}_i^B$ doesn't move on the x axis towards its final destination (which must be the column i or to the left of it), then every label P_j^B with $j > i$ might get obstructed by this movement due to a back-propagating slowdown. For technical reasons apparent in the proof of Lemma 7.4.10, we want to only factor such movements into our bound if they occur *before* $\mathcal{P}_i^B$ meets the leftmost red label $\mathcal{P}_{n_1}^R$.

The general plan is to eventually tie Definition 7.4.7 to the value $\mathcal{V}$ in Definition 7.4.4.

Lemma 7.4.8. *Let t_i be the first time when $\mathcal{P}_i^B$ reaches column i, i.e. an agent with label i arrives at column i for the first time. Then $t_i \geq |front(\mathcal{P}_i^B)| + \mathcal{V}_i^B$.*

Proof Since in the initial configuration $\mathcal{P}_i^B$ sees $i-1$ blue agents in front of it, we have that

$$\left|(\mathcal{P}_i^B)_x - i\right| = \left|front(\mathcal{P}_i^B)\right|$$

(e.g., see Figure 7.7a). Hence $\mathcal{P}_i^B$ requires at least $|front(\mathcal{P}_i^B)|$ horizontal moves to get to column i, and hence requires this many time steps (Observation 7.4.5). If $\mathcal{V}_i^B = 1$ then $\mathcal{P}_i^B$ performs a non-leftward movement before arriving at column i, and this adds one time step. ■

Observation 7.4.9. A given label P^B cannot meet more than one label at any time step.

Proof A given agent A cannot meet more than one agent at any time step, since there are only two rows in the grid (see Figure 7.6). Due to Observation 7.4.5, the same is true of labels. ■

We now proceed to the proof of the main technical lemma of this section.

Lemma 7.4.10. *Let t_{final} be the makespan of ALG on the given initial configuration. Then for any $1 \leq i \leq n_2$,*

$$t_{final} \geq \left|front\left(\mathcal{P}_i^B\left(0\right)\right)\right| + \left|back\left(\mathcal{P}_i^B\left(0\right)\right)\right| + \mathcal{V}_i^B$$

Proof At time t_{final}, P_i^B must be located at or to the left of column i (see Figure 7.7a). Let t_i denote the first time P_i^B reaches column i, and let

$$b_i = |front(\mathcal{P}_i^B)| + \mathcal{V}_i^B.$$

In Lemma 7.4.8 we show that $t_i \geq b_i$. Note that $t_i = b_i$ if and only if $\mathcal{P}_i^B$ moves left at all times $t < t_i$ except at most $\mathcal{V}_i^B$ time steps.

We will prove the lemma by tracking $\mathcal{P}_{n_1}^R$. At time $t = 0$, since the initial configuration is normal, $\mathcal{P}_{n_1}^R$ is the leftmost agent. Hence, between time 0 and t_{final} it ought meet $\mathcal{P}_i^B$. Let M_i be the time of their first meeting, and C_i be the x-coordinate of $\mathcal{P}_i^B$ at time M_i.

We separate the proof into cases.

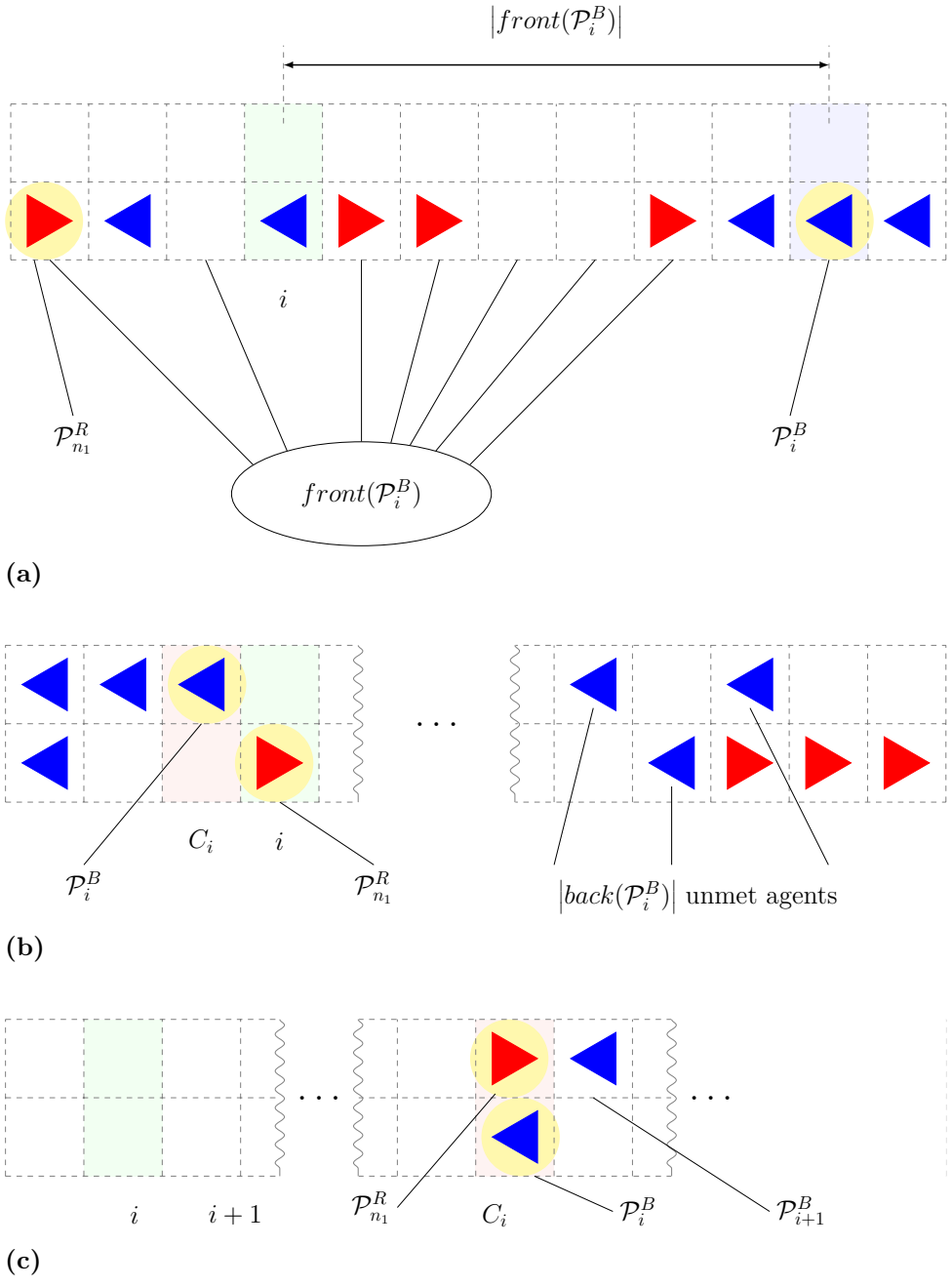
Case 1. Assume $C_i \leq i$. If $C_i \leq i$ then $M_i \geq t_i \geq b_i$. Since in a sorted configuration $\mathcal{P}_{n_1}^R$ must be located to the right of all blue-occupied and empty columns, $\mathcal{P}_{n_1}^R$ must meet the blue labels $\mathcal{P}_{i+1}^B, \mathcal{P}_{i+2}^B, \ldots, \mathcal{P}_{n_2}^B$ before time t_{final}. By Observation 7.4.9, this must take $n_2 - i = |back(\mathcal{P}_i^B)|$ time ticks. Hence in total ALG requires at least $|front(\mathcal{P}_i^B)| + |back(\mathcal{P}_i^B)| + \mathcal{V}_i^B$ time steps to complete, as desired. (See Figure 7.7b)

Case 2. Assume $C_i > i$. In this case, we claim that $\mathcal{P}_{i+1}^B$ cannot reach column C_i before time $M_i + 2$. If column C_i is a mixed column at time M_i then no agent can move into C_i at time $M_i + 1$ (see Figure 7.7c). Otherwise, C_i is occupied by $\mathcal{P}_i^B$, while $\mathcal{P}_{n_1}^R$ occupies $C_i + 1$ in a different row than $\mathcal{P}_i^B$. Since both rows are obstructed by an agent, there is no empty location that will enable $\mathcal{P}_{i+1}^B$ to move into column C_i before time $M_i + 2$.

The label $\mathcal{P}_{i+1}^B$ must reach column $i + 1$ before time t_{final}. By the above, $\mathcal{P}_{i+1}^B$ can reach column C_i for the first time only 2 time steps after $\mathcal{P}_i^B$. The fastest $\mathcal{P}_i^B$ can reach column $i + 1$ is within $b_i - 1$ time steps, and this can only occur when it is able to move left at every time step after it reaches column C_i. Hence the fastest time $\mathcal{P}_{i+1}^B$ can reach column $i + 1$ is $b_i - 1 + 2$.

We can now repeat the argument above for $\mathcal{P}_{i+1}^B$. Note that by Observation 7.4.9, $\mathcal{P}_{i+1}^B$ cannot meet $\mathcal{P}_{n_1}^R$ before time $M_i + 1$. We split the argument into the same two cases:

Figure 7.7 Lemma 7.4.10. (a) At time $t = 0$, $\mathcal{P}_i^B$ is at distance $|front(\mathcal{P})|$ from column i. (b) Case 1: $C_i \leq i$. $\mathcal{P}_{n_1}^R$ must meet at least $|back(\mathcal{P}_B^i)|$ blue agents after meeting $\mathcal{P}_B^i$. (c) Case 2: $C_i > i$. $\mathcal{P}_{i+1}^B$ needs at least two time steps to enter column C_i, and at least $b_i + 1$ time steps to arrive at column $i + 1$.



In Case 1, we assume $\mathcal{P}^B_{i+1}$ reaches column $i+1$ before it meets $\mathcal{P}^R_{n_1}$ or at the same time step. Moreover, $\mathcal{P}^R_{n_1}$ must spend $n_2 - i - 1 = |back(\mathcal{P}^B_i)|-1$ time steps meeting the blue labels $\mathcal{P}^B_{i+2}, \mathcal{P}^B_{i+3}, \ldots, \mathcal{P}^B_{n_2}$. Since $\mathcal{P}^B_{i+1}$ needs at least $b_i - 1 + 2$ time steps to reach column $i+1$, we have that $t_{final} \geq b_i + |back(\mathcal{P}^B_i)|$ just like before, and we are done.

In Case 2, $\mathcal{P}^B_{i+1}$ meets $\mathcal{P}^R_{n_1}$ before reaching column $i+1$. Denote the x-coordinate of $\mathcal{P}^B_{i+1}$ at the time of the meeting by C_{i+1}. By the same argument as before, $\mathcal{P}^B_{i+2}$ can arrive at column C_{i+1} only 2 time steps after $\mathcal{P}^B_{i+1}$. $\mathcal{P}^B_{i+1}$ can arrive at column $i+2$ within $b_i + 1 - 1$ time steps only if it moves left at every time step after meeting $\mathcal{P}^R_{n_1}$. Hence like in the previous argument, the fastest time $\mathcal{P}^B_{i+2}$ can reach column $i+2$ is $b_i + 2$.

We can extend this exact argument by induction to all the blue agents $\mathcal{P}^B_{i+3}$, …, $\mathcal{P}^B_{n_2}$. At every stage of the induction we separate into two cases. In the first case, $\mathcal{P}^B_j$ arrives at column j before it meets $\mathcal{P}^R_{n_1}$ or at the same time step, in which case we can show that $t_{final} \geq b_i + |back(\mathcal{P}^B_i)|$. In the second case, $\mathcal{P}^B_j$ meets $\mathcal{P}^R_{n_1}$ before reaching column j, which implies it needs at least $b_i + (j - i)$ time steps to arrive at column j. By continuing the induction up to $\mathcal{P}^B_{n_2}$ we deduce that the makespan must be at least

$$b_i + (n_2 - i) = b_i + |back(\mathcal{P}^B_i)| = |front(\mathcal{P}^B_i)| + |back(\mathcal{P}^B_i)| + \mathcal{V}^B_i$$

in all cases, so we are done. ■

Lemma 7.4.10 provides a lower bound on the makespan of any sorting algorithm *ALG* with respect to a given initial configuration. The lower bound is based on the *front* and *back*s of the blue agents and on the $\mathcal{V}^B_i$ values induced by *ALG*. By symmetry, a similar argument holds for the red agents, and consequently we may establish:

Corollary 7.2. *Let t_{final} be the makespan of ALG on the given initial configuration. Then*

$$t_{final} \geq \max\left(\max_{1\leq i\leq n_1}(f(P^R_i(0)) + \mathcal{V}^R_i), \max_{1\leq i\leq n_2}(f(P^B_i(0)) + \mathcal{V}^B_i)\right) \tag{7.1}$$

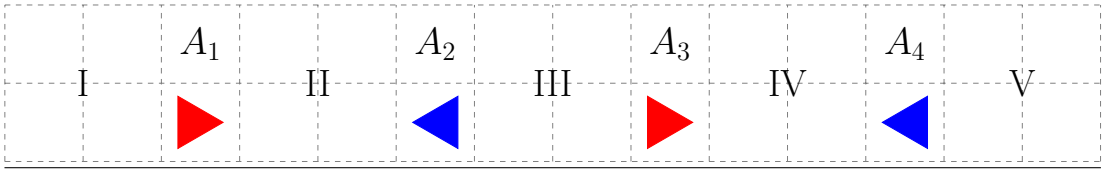
The somewhat unwieldy expression in Corollary 7.2 just says that t_{final} is bounded below by f_{max} plus at most 1, depending on the value of the $\mathcal{V}_i$s. Recalling Definition 7.4.4, we would like to show that the right-hand side of (7.1) is at least as large as $f_{max} + \mathcal{V}$. To show this will require a bit more work.

Lemma 7.4.11. *If a normal initial configuration of agents has both red and blue critical agents, then there must be some blue critical agent to the right of some red critical agent.*

Proof To prove the Lemma we will assume that the claim is wrong and deduce a contradiction.

Suppose there exists an initial configuration with red and blue critical agents, where all the red critical agents are to the right of all the blue critical agents. Let A_2 be some

Figure 7.8 A fictitious configuration with critical agents A_2 (blue) and A_3 (red) not facing each other, per the construction given in Lemma 7.4.11. Column ranges between agents are marked $(I - V)$.



blue critical agent and A_3 be some red critical agent. Let A_1 be the right-most red agent to the left of A_2, and A_4 the left-most blue agent to the right of A_3. We know these agents exist since the initial configuration is normal. Denote the areas between the agents in the following manner: I denotes the set of columns to the left of A_1, II denotes the columns between A_1 and A_2, and so forth until V, which denotes the columns to the right of A_4. This construction is illustrated in Figure 7.8.

Denote by $\mathbf{E}_x, \mathbf{R}_x$ and $\mathbf{B}_x$, respectively, the number of empty, red-occupied, and blue-occupied columns in the set $x \in \{I, II, \ldots, V\}$. We will write down the value of $f(\cdot)$ for all four agents A_i. By definition,

$$f(A_2) = \mathbf{R}_I + \mathbf{E}_I + 1 + \mathbf{E}_{II} + \mathbf{B}_{III} + 1 + \mathbf{B}_V,$$

where the first $+1$ counts A_1, and the second $+1$ counts A_4. Similarly,

$$f(A_3) = \mathbf{R}_I + 1 + \mathbf{R}_{III} + \mathbf{E}_{IV} + 1 + \mathbf{B}_V + \mathbf{E}_V$$

$$f(A_1) = \mathbf{R}_I + \mathbf{B}_{II} + \mathbf{E}_{II} + 1 + \mathbf{B}_{III} + \mathbf{E}_{III} + \mathbf{E}_{IV} + 1 + \mathbf{B}_V + \mathbf{E}_V$$

$$f(A_4) = \mathbf{R}_I + \mathbf{E}_I + 1 + \mathbf{E}_{II} + \mathbf{R}_{III} + \mathbf{E}_{III} + 1 + \mathbf{R}_{IV} + \mathbf{E}_{IV} + \mathbf{B}_V$$

By algebra, we have that if $\mathbf{B}_{III} \geq \mathbf{R}_{III}$ then $f(A_1) \geq f(A_3)$, and otherwise $f(A_4) > f(A_2)$. In the case $f(A_1) \geq f(A_3)$, we have by definition that A_1 is a red critical agent to the left of the blue critical agent A_2, in contradiction to our initial assumption. In the case $f(A_4) > f(A_2)$, we arrive at a contradiction, since $f(A_4)$ cannot exceed $f_{max} = f(A_2)$. This completes the proof. ■

Let us now show that Corollary 7.2 bounds from above our desired lower bound of $f_{max} + \mathcal{V}$, by proving the following:

Lemma 7.4.12.

$$\max\left(\max_{1\leq i\leq n_1}(f(P_i^R(0)) + \mathcal{V}_i^R), \max_{1\leq i\leq n_2}(f(P_i^B(0)) + \mathcal{V}_i^B)\right) \geq f_{max} + \mathcal{V} \tag{7.2}$$

Proof The case where $\mathcal{V} = 0$ is trivial, since the maximum of f taken over all agents is by definition f_{max} and our $\mathcal{V}_i^R, \mathcal{V}_i^B$s are non-negative. So let us assume $\mathcal{V} = 1$. By definition, this means either (a) there is a critical agent A that sees another agent

immediately in front of it in the initial configuration, or (b) there are two critical agents of different colors in the initial configuration.

In case (a), let us suppose without loss of generality that A is red, and let $ord(A) = i$. Then necessarily $f(P_i^R(0)) = f_{max}$ and, since A cannot move horizontally nor change labels in the first time step of ALG, $\mathcal{V}_i^R = 1$. Hence $f(P_i^R(0)) + \mathcal{V}_i^R = f_{max} + \mathcal{V}$ which proves Inequality (7.2).

In case (b), Lemma 7.4.11 tells us that there exists a pair of *red and blue critical agents* A^R and A^B facing each other. Let $ord(A^R) = i$ and $ord(A^B) = j$. If A^R and A^B do not change labels before meeting (i.e., for all times t prior to the meeting, $ord(A^R, t) = i$ and $ord(A^B, t) = j$), then necessarily one of the labels $\mathcal{P}_i^R$ or $\mathcal{P}_j^B$ must move to the upper row so that they can bypass each other. Hence $\mathcal{V}_i^R = 1$ or $\mathcal{V}_j^B = 1$, and similar to case (a), this establishes Inequality (7.2).

Otherwise, either A^R or A^B changes their label before meeting. Let us suppose wlog that A^R changes its label at time t_0, i.e. $ord(A^R, t_0 - 1) = i \wedge ord(A^R, t_0) \neq i$. At t_0, A^R's label can be either $i+1$ or $i-1$. It is straightforward to verify by hand that in either case, $\mathcal{V}_i^R = 1$, since in order for two adjacent agents to swap labels, one of them must wait or move toward the other, and this causes $\mathcal{V}^R$ to become 1 for both agents that swapped labels. Hence, similar to case (a) and (b), this establishes Inequality (7.2). ■

Corollary 7.2 and Lemma 7.4.12 together establish the lower bound $f_{max} + \mathcal{V}$ of Theorem 7.1. It remains to show an algorithm that achieves this bound exactly.

7.5 Optimal algorithm for normal configurations

In this section we present a simple agent sorting algorithm. We show that despite its simplicity, the algorithm is optimal for normal initial configurations, in the sense that its makespan **always** meets the lower bound established in Theorem 7.1, thus completing the proof of Theorem 7.1.

Algorithm 7.1 is simple to describe: *aim to move in your desired direction*, left for blue agents and right for red agents. The only twist is the first tick, where either all red or all blue agents move to the second row. Which color should we move vertically? The answer comes from Theorem 7.1; we want to avoid paying more than $f_{max} + \mathcal{V}$ time steps by moving vertically only the color without critical agents. This enables critical agents to move horizontally without being delayed by a vertical movement. If there are red critical agents - we move blue agents to the second row, otherwise red agents are moved.

Denote the above algorithm ALG_1. To establish its makespan, let us first study the simple scenario where all blue agents are in the one row and all red agents are in the other, and at every time step they simply move horizontally in the desired direction:

Algorithm 7.1 Optimal sorting algorithm for unsorted normal initial configurations.

init $t \leftarrow 0$
for each agent $A \in \mathcal{A}$ **do**
    calculate $f(A)$ to determine the critical agents
end for
if there are critical agents of both colors **then**
    $c \leftarrow$ red
else
    $c \leftarrow$ the color of critical agents
end if
for each agent $A \in \mathcal{A}$ **do**
    if A's color is c **then**
        move one step in the desired direction if the adjacent location in that direction is not occupied at the beginning of time $t = 0$
    else
        move to the second row
    end if
end for
while configuration is not sorted **do**
    $t \leftarrow t + 1$
    for each red agent $\in \mathcal{A}$ **do**
        if right-adjacent location is empty at the beginning of t **then**
            move one step right
        end if
    end for
    for each blue agent $\in \mathcal{A}$ **do**
        if left-adjacent location is empty at the beginning of t **then**
            move one step left
        end if
    end for
end while

Lemma 7.5.1. *Consider a normal configuration where all red agents are at the top row and all blue agents are at the bottom row, and every column contains at most one agent.*

Let $ALG_1^{\perp}$ denote the algorithm that says: at every time step, red agents move right unless the adjacent location to their right is occupied, and blue agents move left unless the adjacent location to their left is occupied. Then the makespan of $ALG_1^{\perp}$ on this configuration is at most f_{max}.

Proof For any red agent R, we define $front^{\perp}(R,t)$ as the set of all empty locations on R's row which are to the right of R at time t. Furthermore, define $back^{\perp}(R,t)$ to be the set of all red agents to the left of R.

Similarly, for any blue agent B, we define $front^{\perp}(B,t)$ as the set of all empty locations on B's row which are to the left of B at time t, and define $back^{\perp}(B,t)$ to be the set of all blue agents to the right of B.

For every agent A define $f^{\perp}(A,t)$ at time t in the following manner:

- if $front^{\perp}(A,t)$ contains no empty locations, $f^{\perp}(A,t) = 0$
- otherwise, $f^{\perp}(A,t) = |front^{\perp}(A,t)| + |back^{\perp}(A,t)|$

In the Lemma's assumed agent configuration, $|front^{\perp}(A,0)| > 0$ for any agent A. Hence for all agents $f^{\perp}(A,0) \neq 0$. Recalling Definition 7.4.2, this means $f^{\perp}(A,0) = f(A)$ for any agent A. Hence, there is a critical agent A for which $f^{\perp}(A,0) = f_{max}$.

Let us define $f^{\perp}_{max}(t) = \max_{A \in \mathcal{A}} f^{\perp}(A,t)$. By the above, $f^{\perp}_{max}(0) = f_{max}$. When $f^{\perp}_{max}(t) = 0$, the agent configuration is sorted, since every agent has moved as far as it could in its desired direction. We will show that $f^{\perp}_{max}(t)$ decreases every time step as long as it is not 0, which completes the proof.

At time t, let A^* be an agent for which $f^{\perp}(A^*,t) = f^{\perp}_{max}(t)$. Let us suppose w.l.o.g. that A^* is a red agent. Note that if a red agent R is an adjacent rightward neighbor of A^* at time t then

(a) if $f^{\perp}(R,t) = 0$ then $f^{\perp}(A^*,t) = 0$, and

(b) $f^{\perp}(A^*,t) = f^{\perp}(R,t) - 1$ otherwise.

(b) is of course impossible, since A^* maximizes $f^{\perp}(\cdot,t)$. Hence, if A^* cannot move right at time t, we have that $f^{\perp}(A^*,t) = f_{max}(t) = 0$, so the configuration is sorted. Otherwise, A^* sees an empty location immediately to its right, and it moves to it at time t. This causes $f^{\perp}(A^*,t)$ to decrease by 1.

Since the above argument is true for *any* agent that maximizes $f^{\perp}(\cdot,t)$, we see that as long as $f^{\perp}_{max}(t) \neq 0$, f_{max} decreases at every time step. ■

An immediate corollary of Lemma 7.5.1, which (as mentioned in the introduction) relates to asymmetric simple exclusion processes and may be of independent interest, is:

Corollary 7.3. *Assume j agents located on a single row (without access to a spare row) move right at every time step where they are unobstructed by another agent. Then after exactly f_{max} time steps, the j right-most columns are occupied by an agent.*

Theorem 7.4. *For a given initial normal configuration, ALG_1's makespan is $f_{max}+\mathcal{V}$.*

Proof By Theorem 7.1, we know the makespan is at least $f_{max} + \mathcal{V}$. Let us show it is at most this.

Let us define f'_{max} to be calculated like f_{max} in Definition 7.4.2, but over the agent configuration at time $t = 1$. Note that $f_{max} \geq f'_{max}$, since ALG_1 never moves agents in a way that can increase this value.

Note that from time $t = 1$ onwards, all agents are on two separate rows and execute $ALG_1^{\perp}$ (of Lemma 7.5.1) in their respective row. Hence, by Lemma 7.5.1, our algorithm's makespan is at most $f'_{max} + 1 \leq f_{max} + 1$. Hence, if $\mathcal{V} = 1$, we are done.

Let us deal with the case $\mathcal{V} = 0$. We assume, w.l.o.g., that there are red critical agents. Since $\mathcal{V} = 0$, this means there are no blue critical agents, and also that every red critical agent takes a step to the right in the first time step of the algorithm. Since the maximum value of f is obtained over a red agent, we know that

$$\max_{B \in \mathcal{B}} f(B) \leq f_{max} - 1.$$

Furthermore, since every red critical agent moves right at time $t = 0$, when we re-calculate f over the configuration at time $t = 1$, f_{max} will have decreased by 1 (since every critical agent A has shifted an empty location from $front(A)$ to $back(A)$). Hence $f'_{max} = f_{max} - 1$, and so our makespan is at most f_{max}, as claimed. ■

Theorem 7.4 and the lower bound result of the previous section establish Theorem 7.1. We see that ALG_1 is an optimal algorithm for physically sorting any normal initial configuration. In the next section, we will extend this result to non-normal configurations.

7.6 Non-normal initial configurations

In the previous sections we have handled a subset of all the possible initial configurations - the normal configurations (Definition 7.3.1). In this section, we extend our previously derived lower bound to all possible initial configurations, and extend our optimal algorithm to such configurations.

Definition 7.6.1. Consider the configuration of agents at time t. Let R be the leftmost red agent and let B be the rightmost blue agent. The **maximal normal sub-configuration at time** t is the set of all columns in the interval $[R_x(t), B_x(t)]$. Note that this set is empty when $R_x(t) > B_x(t)$.

Furthermore, let $\mathcal{S}$ be the (possibly empty) maximal normal sub-configuration at time $t = 0$. Let $\mathcal{S}^c = \mathcal{A} \setminus \mathcal{S}$ be the set of columns outside $\mathcal{S}$.

We define a function f^* to equal f for all agents in (the columns of) $\mathcal{S}$, and for any agent A in $\mathcal{S}^c$ we set $f^*(A) = |front(A)|$. We further define $f^*_{max} = \max_{A \in \mathcal{A}} f^*(A)$.

Definition 7.6.2. An agent $A \in \mathcal{A}$ for which $f^*(A) = f^*_{max}$ is called an f^***-critical agent**.

We will define $\mathcal{V}^*$ to closely resemble $\mathcal{V}$:

Definition 7.6.3. $\mathcal{V}^*$ will equal 1 if:

1. There is both a red and a blue f^*-critical agent **and** $\mathcal{S} \neq \varnothing$, or
2. There is an f^*-critical agent such that, in the initial configuration, another agent is located immediately in front of it **and** $f^*_{max} > 0$.

Otherwise, $\mathcal{V}^* = 0$.

Lemma 7.4.10 can now be extended to non-normal initial configurations as follows:

Lemma 7.6.4. *The makespan of any algorithm that brings the initial configuration to a sorted configuration is at least* $f^*_{max} + \mathcal{V}^*$.

Proof The argument is a rather straightforward generalization of the previous sections.

We assume first that $\mathcal{S} \neq \varnothing$. We note that in this case, agents outside $\mathcal{S}$ are not f^*-critical. Indeed, for any same-colored agents $A_1 \in \mathcal{S}$ and $A_2 \notin \mathcal{S}$, $front(A_2) \subsetneq front(A_1)$. Therefore, we trivially have that $f^*(A_2) < f^*(A_1)$. We note that the argument in Lemma 7.4.10 generalizes genuinely to agents inside $\mathcal{S}$. Consequently, Lemma 7.4.10 provides a lower bound on makespan, since $f \equiv f^*$ on $\mathcal{S}$ and when restricted to this set, $\mathcal{V}^*$ is equivalent to the definition of $\mathcal{V}$.

Now let us assume $\mathcal{S} = \varnothing$, i.e. in the initial configuration no pair of red and blue agents face each other. Each label $\mathcal{P}^B_i$ must traverse at least $|front(\mathcal{P}^B_i(0))|$ columns before a sorted configuration can be reached and the same claim is true for $\mathcal{P}^R_i$. In particular, by definition, f^*-critical agents will need to traverse f^*_{max} columns to reach their position. If $\mathcal{V}^* = 0$, this establishes our desired lower bound.

If $\mathcal{V}^* = 1$, we have that there is an f^*-critical agent that is blocked by an adjacent agent, hence cannot change columns in the first time step. Furthermore, since $f^*_{max} > 0$, the configuration is not sorted, so this agent *must* traverse at least one column. Hence, at least one f^*-critical agent must spend $\mathcal{V}^*$ ticks before starting to move horizontally, establishing a lower bound of $f^*_{max} + \mathcal{V}^*$ time steps before a sorted configuration is reached. ■

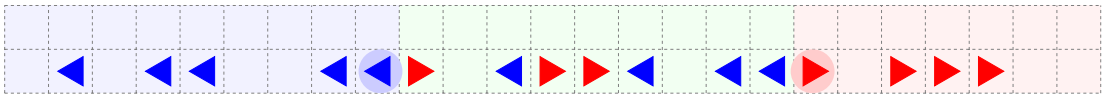
We are now interested in designing an optimal physical sorting algorithm for non-normal configurations - Algorithm 7.2. The algorithm we present will work as follows: the agents that are inside S at time $t = 0$ will continue to execute Algorithm 7.1 as before, whereas all agents initialized outside $\mathcal{S}$ will execute an "alternating row split"

strategy. These strategies are executed independently by the two sets of agents: agents initialized in $\mathcal{S}$ will not reach the $\mathcal{S}^c$ columns fast enough to interact with agents initialized outside $\mathcal{S}^c$.

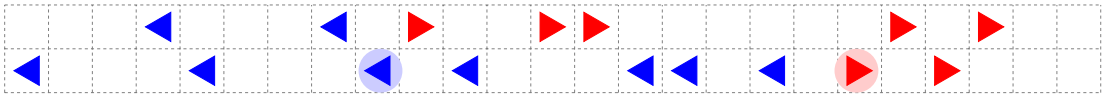
Definition 7.6.5. Let r_0 be the label of the leftmost red agent in $\mathcal{S}^c$ and b_0 be the label of the rightmost blue agent in $\mathcal{S}^c$ at time $t = 0$. $\mathcal{P}^R_{r_0}(0)$ and $\mathcal{P}^B_{b_0}(0)$ will be called red and blue **anchor** agents respectively.

The "alternating row split" strategy is based on the following simple idea: at time $t = 0$, split agents outside $\mathcal{S}$ in an alternating fashion between the two rows, so that there is at least one empty space between each pair of same-colored agents in the same row (see Figure 7.9). Once this is done, each agent in $\mathcal{S}^c$ can move horizontally in its desired direction at every time step without any further delays - see Figure 7.9.

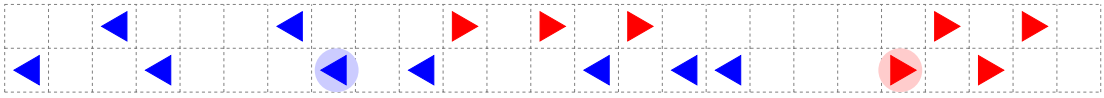
Figure 7.9 We illustrate 5 time steps of a run of Algorithm 7.2 on a non-normal initial configuration. The maximal normal sub-configuration $\mathcal{S}$ is highlighted in green. The left-hand and right-hand sides of $\mathcal{S}^c$ are highlighted in blue and red respectively. The blue $\mathcal{P}^B_{b_0}$ and red $\mathcal{P}^R_{r_0}$ **anchor agents** are circled in blue and red respectively.



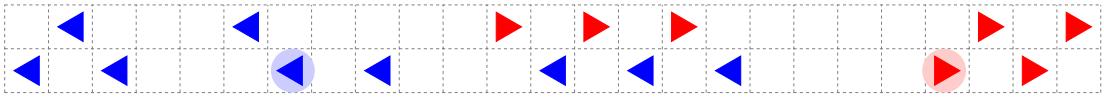
(a) $t = 0$



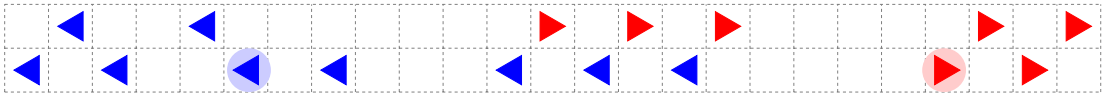
(b) $t = 1$



(c) $t = 2$



(d) $t = 3$



(e) $t = 4$

Algorithm 7.2 Optimal sorting algorithm for any unsorted initial configuration.

init $t \leftarrow 0$, $\mathcal{Q} \leftarrow$ all agents initialized in the columns of $\mathcal{S}$, $\mathcal{P} \leftarrow$ all agents initialized in the columns of $\mathcal{S}^c$
for each agent $A \in \mathcal{A}$ **do**
    calculate $f^*(A)$ to determine the critical agents
    if there are f^*-critical agents of both colors **then**
        $c \leftarrow$ red
    else
        $c \leftarrow$ the color of the f^*-critical agents
    end if
    move all agents in $\mathcal{Q}$ whose color is not c to the second row
end for
for each agent $A \in \mathcal{P}$ **do**
    if A is red **and** $(r_0 - ord(A, 0))$ is odd **then**
        move A to the second row
    else if A is blue **and** $(b_0 - ord(A, 0))$ is odd **then**
        move A to the second row
    end if
end for
for each agent $A \in \mathcal{A}$ **do**
    if A has not moved due to a previous step of the algorithm **then**
        move one step in the desired direction (left for blue agents, right for red agents) if the adjacent location in that direction is not occupied at the beginning of time $t = 0$
    end if
end for
while the configuration is not sorted **do**
    $t \leftarrow t + 1$
    for each red agent A **do**
        if there is a mixed, empty, or blue-occupied column to the right of A at the beginning of time t **then**
            move one step right
        end if
    end for
    for each blue agent A **do**
        if there is a mixed, empty, or red-occupied column to the left of B at the beginning of time t **then**
            move one step left
        end if
    end for
end while

We will show that Algorithm 7.2 is an optimal sorting algorithm.

Lemma 7.6.6. *If $\mathcal{S} = \varnothing$, the makespan of Algorithm 7.2 is exactly*

$$f^*_{max} + \mathcal{V}^*$$

Proof Lemma 7.6.4 shows $f^*_{max} + \mathcal{V}^*$ is a lower bound on the makespan, so we just need to prove that it is also an upper bound.

If $\mathcal{S} = \varnothing$, then at time $t = 0$ all the agents split between the two rows in an alternating fashion (see Figure 7.9). In every subsequent time step any agent that has not reached its final sorted position moves horizontally along its row. Thus, each agent $A \in \mathcal{S}^c$ takes at most $front(A) + 1 \leq f^*(A) + 1$ time steps to reach its final position in the sorted configuration.

We split the proof into cases.

Case 1: There are only red f^*-critical agents. Recalling Definition 7.6.5, we infer that the red anchor agent is necessarily f^*-critical. Denote this agent W.

Suppose W moves horizontally (i.e., rightwards) at time $t = 0$. It will continue to do so at every subsequent time-step, thus reaching its desired column in $front(W) = f^*(W)$ time steps. Note that if W moves horizontally at time $t = 0$ according to Algorithm 7.2, it is necessarily the **only** red f^*-critical agent, because it has an empty space in front of it that no other agent counts towards its f^* value. Thus in this case, $\mathcal{V}^* = 0$.

Suppose on the other hand that W doesn't move horizontally at time $t = 0$. Then, since W is a critical agent, $\mathcal{V}^* = 1$, and W will reach its desired column in $front(W) + 1 = f^*(W) + \mathcal{V}^*$ time steps.

Every other agent A reaches its desired column in at most $f^*(A) + 1 \leq f^*(W) + \mathcal{V}^*$ time steps. Thus Algorithm 7.2's makespan is at most $f^*_{max} + \mathcal{V}^*$.

Case 2: There are only blue f^*-critical agents. This is the same as case 1.

Case 3: There are both red and blue f^*-critical agents. Since when $\mathcal{S} = \varnothing$, red and blue agents both move independently on disjoint sets of columns, thus we may combine the arguments of Case 1 and Case 2 to infer that the makespan is $f^*_{max} + \mathcal{V}^*$. ■

Observation 7.6.7. If $\mathcal{S} \neq \varnothing$ then any f^*-critical agent is necessarily in a column of S at time $t = 0$.

Proof Suppose there is an f^*-critical agent W in a column of $\mathcal{S}^c$. Assume without loss of generality that W is red. Since $\mathcal{S} \neq \varnothing$, there is necessarily a red agent W' in a column of $\mathcal{S}$ with smaller x-coordinate than W. We have that $|front(W')| > |front(W)|$, since everything that is in front of W is also in front of W', but $front(W')$ contains a blue-occupied column between W and W' which is not in $front(W')$. By definition $f^*(W') \geq |front(W')| > |front(W)| = f^*(W)$, a contradiction to the assumption that W is a critical agent. ■

Proposition 7.6.8. *In every (normal or non-normal) configuration, Algorithm 7.2 completes in exactly*

$$f^*_{max} + \mathcal{V}^*$$

time steps.

Proof If $\mathcal{S} = \varnothing$, the claim follows from Lemma 7.6.6. We assume therefore that $\mathcal{S} \neq \varnothing$.

Note that at every time step starting at $t = 1$ all agents execute the same logic: red agents move right and blue agents move left whenever they are unobstructed.

In Algorithm 7.2, after time $t = 0$, any blue or red agent initialized in the columns of $\mathcal{S}^c$ can move in its desired direction until it settles in its final column in the physical sorting. Furthermore, agents initialized in the columns of $\mathcal{S}$ need at least three time ticks to reach any column of $\mathcal{S}^c$ (see Figure 7.9). Therefore, agents initialized in $\mathcal{S}$'s columns can never catch up with agents initialized in $\mathcal{S}^c$ before the $\mathcal{S}^c$ agents reach their final sorted position. In other words, agents in $\mathcal{S}^c$ never obstruct agents in $\mathcal{S}$. Moreover, the agents initialized in $\mathcal{S}$ arrive at their final sorted position strictly later than agents in $\mathcal{S}^c$.

Let r be the number of red agents in the columns of $\mathcal{S}^c$. Note that no agent initialized in $\mathcal{S}$ will ever enter the rightmost r columns, since these will be occupied by red $\mathcal{S}^c$ agents and Algorithm 7.2 does not let two red agents occupy the same column at any time step. Analogously, let b be the number of blue agents in the columns of $\mathcal{S}^c$. Then no agent initialized in $\mathcal{S}$ will ever enter the leftmost b columns.

Let $\mathcal{C}^*$ be our initial agent configuration, and let $\mathcal{C}$ be the initial configuration where we delete all agents in the $\mathcal{S}^c$ columns and delete the rightmost r columns and the leftmost b columns. It is not difficult to show from the previous two paragraphs that the makespan of Algorithm 7.2 on $\mathcal{C}^*$ is the same as the makespan of Algorithm 7.1 on $\mathcal{C}$.

According to Observation 7.6.7, all critical agents in the configuration $\mathcal{C}^*$ are initialized in $\mathcal{S}$. Let A be any red agent initialized in $\mathcal{S}$. Every red agent that we remove from $\mathcal{C}^*$ to create $\mathcal{C}$ (i.e., red agents in $\mathcal{C}^* \setminus \mathcal{C}$) increases $|front(A)|$ by 1. Every column we remove from the right-hand side decreases $|front(A)|$ by 1 and offsets this. Thus the value of $|front(A)|$ does not change between $\mathcal{C}$ and $\mathcal{C}^*$. The size of $front$ similarly remains unchanged for every blue agent initialized in $\mathcal{S}$. $|back(A)|$ remains unchanged for red and blue agents by definition.

Consequently, $f^*_{max} + \mathcal{V}^*$ over $\mathcal{C}^*$ is equal to $f_{max} + \mathcal{V}$ over $\mathcal{C}$. Thus the makespan of Algorithm 7.2 over $\mathcal{C}^*$ is $f^*_{max} + \mathcal{V}^*$. ■

In conclusion, the makespan of any sorting algorithm over any initial configuration is at least $f^*_{max} + \mathcal{V}^*$. This lower bound is precise; Algorithm 7.2 meets it exactly. This generalizes Theorem 7.1 to non-normal configurations.

Theorem 7.5. *The makespan of any sorting algorithm over a given (normal or non-normal) initial configuration is at least* $f^*_{max} + \mathcal{V}^*$*. This lower bound is precise; Algorithm 7.2 meets it exactly.*

7.7 Discussion

We studied the problem of sorting vehicles or "agents" on a horizontal two-row highway, sending red vehicles to the right and blue vehicles to the left as quickly as possible. We derived an exact lower bound for the amount of time it takes an arbitrary configuration of such vehicles starting at the bottom row to arrive at a sorted configuration, and presented an optimal sorting algorithm that attains this lower bound.

The instance-optimal algorithm we presented for sorting normal configurations (Algorithm 7.1) requires global knowledge of the initial agent configuration to compute f_{max}, as the value of f_{max} determines which color of agent it raises to the upper row. Consider the algorithm that, instead of computing f_{max}, simply raises all the *blue* agents to the upper row, and otherwise proceeds the same as Algorithm 7.1. This is a straightforward algorithm that requires no global knowledge; in fact, it can be implemented by decentralized agents with local sensing and no communication (see Section 7.7.1, Algorithm 7.3). Theorem 7.1 shows that this algorithm is at most 1 time step slower than Algorithm 7.1. Figure 7.10 in Section 7.7.1 shows an example run of both algorithms. We find it significant that the instance-optimal strategy for this problem can be approximated by a very simple, decentralized and local sensing-based algorithm.

It is not clear whether a simple decentralized strategy exists for non-normal configurations. In our general sorting algorithm (Algorithm 7.2), agents must know whether they are in the maximal subnormal configuration or outside of it to determine their movements, and so a local decentralized algorithm with nearly identical performance is more difficult to conceive of.

In future work, it will be interesting to consider the following extensions of our problem:

1. Settings where there are more than two rows which the agents may use, and where agents are initialized on arbitrary rows and columns.

2. General permutations. Suppose that instead of just 2 colors, there are k different colors. What is the optimal way to sort the agents, such that all agents of color i are between those of color $i-1$ and $i+1$?

3. Agents moving at different velocities, e.g., a grid square every 2 or 4 time steps instead of 1.

Physical sorting problems such as the one described in this chapter force algorithm designers to take into account a number of factors that are not present in more traditional settings, such as physical motion and collision avoidance. When compared to

traditional combinatorial algorithms, we believe that there is a lot of room for further theoretical developments in this domain, as even results that at a glance might seem straightforward currently require specialized analysis.

7.7.1 An almost optimal distributed solution

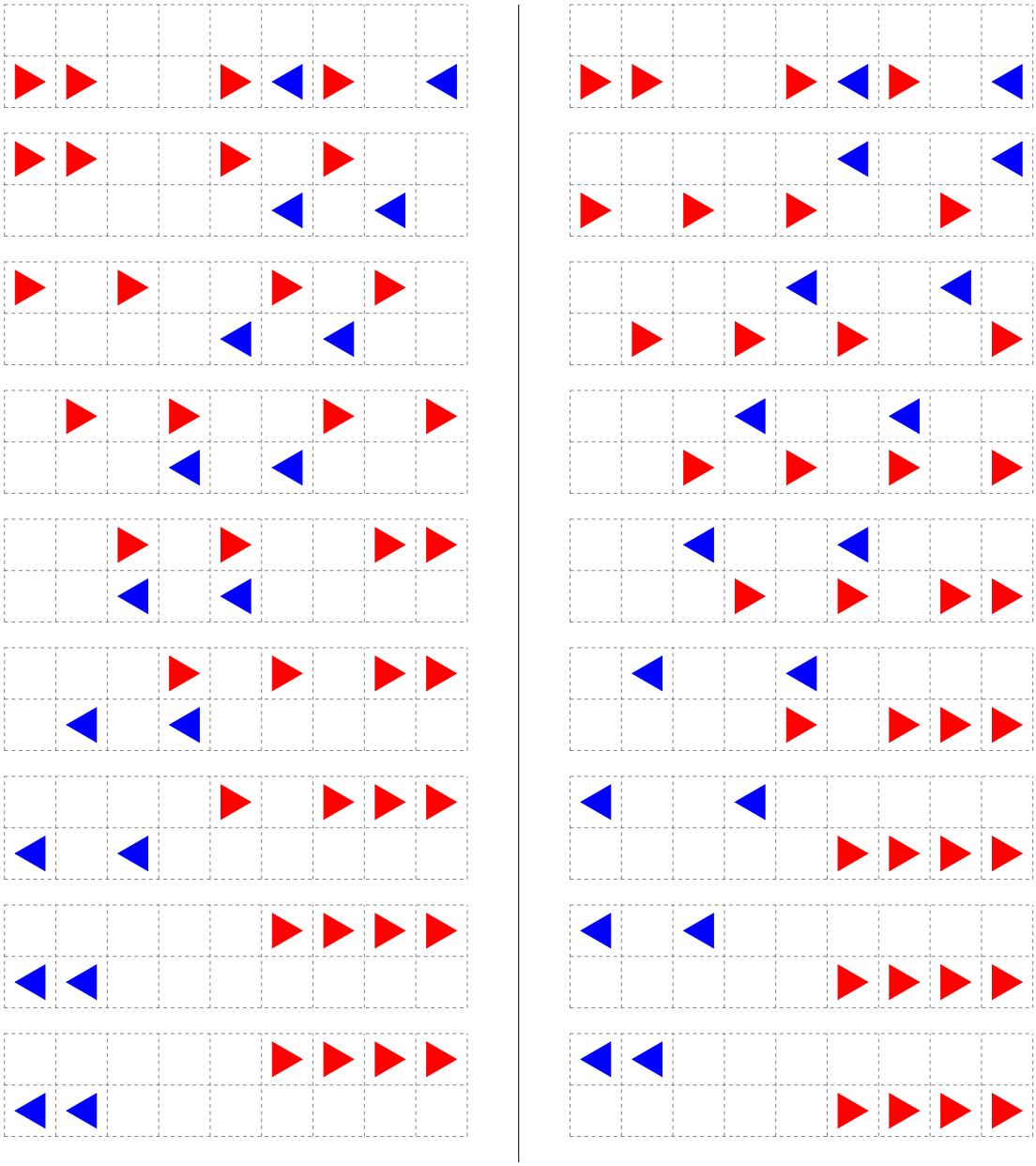
The proposed centralized solution can be easily parallelized to produce an almost trivial decentralized algorithm, that, nonetheless, sorts agents in only a single tick slower than the instance-optimal Algorithm 7.1. Every agent A has a memory bit d which is used to track whether the current time step is 0 or not. At time 0, all the blue agents move to the upper row. In Algorithm 7.3, we describe what each (decentralized) mobile agent does at every time tick t:

Algorithm 7.3 Almost optimal decentralized sorting algorithm for an unsorted initial normal configuration.

```
init memory bit d ← 0
if I am a blue agent then
    if d = 0 then
        move to the second row
    else if adjacent location to the left is empty then
        move one step left
    end if
else if adjacent location to the right is empty then
    move one step right
end if
d ← 1
```

Based on the analysis of Theorem 7.1, this algorithm is at most 1 time step slower than the optimal time for any given initial configuration. The additional time tick is incurred because in an instance-optimal solution, sometimes we must raise the red agents to the second row rather than the blue agents. In Figure 7.10, we provide an example run comparing Algorithm 7.1 and Algorithm 7.3.

Figure 7.10 A side-by-side algorithm execution on a given initial grid configuration. (Left) centralized Algorithm 7.1, (right) distributed Algorithm 7.3. The centralized algorithm finishes 1 time step before the decentralized algorithm ($t = 7$ and $t = 8$ respectively).



Chapter 8

Conclusion

Life tells a similar story whether viewed at the level of the microorganism (an ant, a germ, a virus) or at the level of the macroorganism (a family, a company, a culture). In this story organisms evolve local rules of behavior that help them grow, compete, cooperate, make and unmake. The best and most resilient rules propagate through time and define our world's future. The topic of this thesis was understanding and designing local rules of behavior that are both natural and effective. Under the paradigms of multi-agent systems on graphs and ant-like swarm robotics we studied two mathematical models of social insects, algorithms for swarm deployment in unknown environments, and the traffic management-related problem of physical sorting. We hope that the results we obtained in their totality make a convincing argument that the local, precise and expressive language of swarm-robotic algorithms is an important vehicle for exploring what makes Mother Nature's rules so resilient and successful. We hope further that the models, techniques, and ideas in this thesis serve as a useful starting point for any reader interested in the mathematical analysis of multi-agent systems on graphs. We believe that the bag of mathematical tools we collected is useful beyond just the scope of this dissertation, and has potential applications to the analysis of many more types of multi-agent systems.

We would like to finish by briefly discussing two manuscripts that did not make it into the main body of this work.

Stigmergy-based, Dual-Layer Coverage of Unknown Indoor Regions. In the work [RAB22] we expand on the topic of Chapter 6 and present several algorithms for uniformly covering an unknown indoor region with a two-layered swarm of mobile agents. Same as Chapter 6, these algorithms draw upon the meta-concept of "stigmergy" - communication via the environment - by having robots that settle inside the region become part of the environment in the form of beacons that guide mobile robots to as-yet unexplored locations.

The algorithms presented in [RAB22] improve upon the DFS-based algorithm of Chapter 6 by introducing a backward propagating information diffusion process through

which settled agents are able to prevent mobile agents from entering already-explored areas, redirecting them to more fruitful avenues of search instead. Unlike Chapter 6, and unlike many other works in this area, we consider the requirement of informing an outside operator with limited information that the coverage mission is complete. Even with this additional requirement we show, both through simulations and mathematical proofs, that the dual role concept results in linear-time termination, while also besting well-known algorithms in the literature (such as Hsiang et al.'s BFLF and DFLF [HAB+04]) in terms of energy use.

Multi-Agent Distributed and Decentralized Geometric Task Allocation. In the work [AKB+22] we explore the topic of dispersing a robotic swarm of autonomous mobile agents over a region to find and fulfill an *a priori unknown* set of tasks. The location of the tasks and the number of agents required to complete them are not given to the agents in advance, and may even change over time. The goal of the agents is to explore the environment to find the tasks, and to position themselves in the region based on the requirements of each task. The agents must also relocate in response to changes in the set of tasks - for example, agents that complete a given task should go on to help other agents complete their tasks. Examples of this kind of setting include search and rescue missions, where agents must find and assist an unknown number of people, or forest fires, where the spread and intensity of fire evolves over time and requires varying numbers of firefighting drones to cover.

Unlike the works that we explored in this thesis, the setting of [AKB+22] is not graphs: we assume the agents' topology is some closed subregion $\mathcal{L}$ of the plane $\mathbb{R}^2$ (such as the unit square $\mathcal{L} = [0,1] \times [0,1]$) within which they are able to move about freely, and that the agents' tasks are represented by an *a priori unknown* scalar field $\Phi(x, y)$ that determines how many agents are needed in each location. Fundamentally, however, this is still a work about swarm robotics and the ants paradigm - the agents are autonomous, oblivious and indistinguishable, and have finite sensing range. Swarm robotics is uniquely positioned to handle task allocation in unknown environments, because the agents can quickly cover a very large area to locate the tasks, and because there are enough agents that we need not worry about some of the agents not finding a task to work on.

In [AKB+22] we propose to solve task allocation problems through *attraction-repulsion dynamics*, wherein agents repulse each other and are attracted to locations in the region determined by Φ. The heuristic behind this idea is that the repulsive forces agents affect each other with cause the swarm to expand uniformly, thus covering the region of interest $\mathcal{L}$ and discovering the values of Φ. Attractive forces, on the other hand, cause agents to accumulate according to tasks' requirements.

Figure 8.1 shows a task allocation scenario where agents, starting in the middle of the region, search for tasks placed randomly within the region through attraction-repulsion dynamics. Each task demands a different number of agents to complete.

When a sufficient number of agents reach a task, the task is completed and removed from Φ, freeing agents to move to other tasks. Figure 8.2 shows a task allocation where tasks (not drawn) are static and placed in the shape of a plane, causing the agents to enter said formation and demonstrating the precision and scalability of our approach.

Figure 8.1 Simulation of a task allocation scenario where tasks disappear upon reaching the demanded number of agents. Agents move according to attraction-repulsion dynamics. The current time step is written in the top left corner of each frame. Agents' sensing range is depicted by the transparent gray disk (sensing range is depicted for a single agent to prevent visual clutter). 200 agents begin at the center of the region and expand outward, finding targets. The total agent demand of the targets is 204.

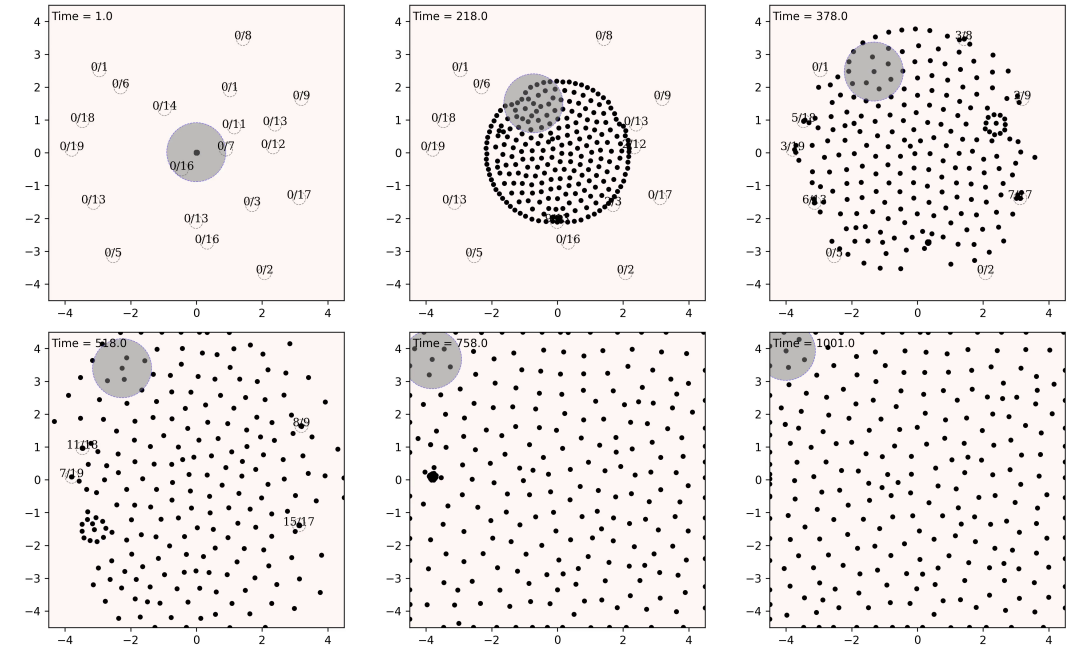
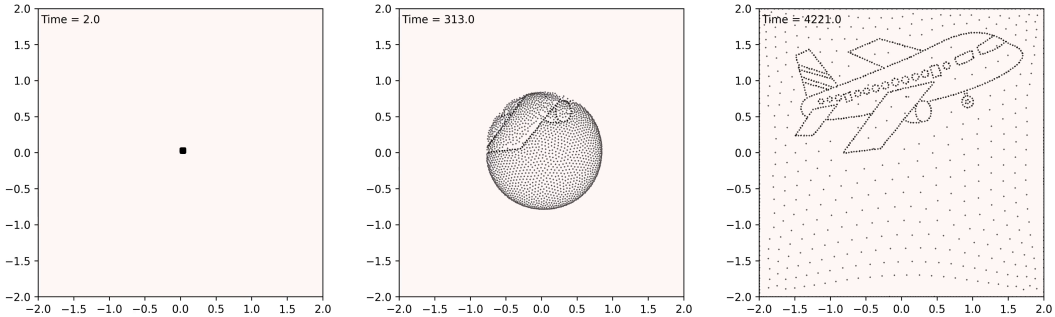


Figure 8.2 Simulation of a task allocation scenario with targets placed in a plane-shaped configuration. The total agent demand of the targets is 1240, and the number of agents is 2000. Targets not drawn for the sake of visual clarity. Agent sensing radius, not depicted in the image, is 1/10th of the bounding box.



Bibliography

[AA15] Gil Ariel and Amir Ayali. Locust collective motion and its modeling. *PLOS Computational Biology*, 11(12):e1004522, December 2015. Frederick R. Adler, editor. URL: https://doi.org/10.1371/journal.pcbi.1004522.

[AAA16] Guy Amichay, Gil Ariel, and Amir Ayali. The effect of changing topography on the coordinated marching of locust nymphs. *PeerJ*, 4:e2742, December 2016. URL: https://doi.org/10.7717/peerj.2742.

[AAB21] Michael Amir, Noa Agmon, and Alfred M Bruckstein. A discrete model of collective marching on rings. *International Symposium Distributed Autonomous Robotic Systems*:320–334, 2021.

[AAB22] Michael Amir, Noa Agmon, and Alfred M. Bruckstein. A locust-inspired model of collective marching on rings. *Entropy*, 24(7):918, June 2022. URL: http://dx.doi.org/10.3390/e24070918.

[AB19a] Michael Amir and Alfred M Bruckstein. Probabilistic pursuits on graphs. *Theoretical Computer Science*, 2019.

[AB19b] Michael Amir and Alfred M. Bruckstein. Minimizing Travel in the Uniform Dispersal Problem for Robotic Sensors. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 113–121, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems, 2019. (Visited on 07/30/2022).

[AB20] Michael Amir and Alfred M. Bruckstein. Fast uniform dispersion of a crash-prone swarm. In *Proceedings of Robotics: Science and Systems*, RSS '20, 2020.

[ABC17] Peyman Afshani, Jérémy Barbay, and Timothy M Chan. Instance-optimal geometric algorithms. *Journal of the ACM (JACM)*, 64(1):1–38, 2017.

[AF95] David Aldous and James Fill. Reversible markov chains and random walks on graphs, 1995.

[AHK06] Noa Agmon, Noam Hazon, and Gal A Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* Pages 1698–1703. IEEE, 2006.

[AHKG+08] Noa Agmon, Noam Hazon, Gal A Kaminka, MAVERICK Group, et al. The giving tree: constructing trees for efficient offline and online multi-robot coverage. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):143–168, 2008.

[AKB+22] Michael Amir, Yigal Koifman, Yakov Bloch, Ariel Barel, and Alfred M. Bruckstein. Multi-agent distributed and decentralized geometric task allocation, 2022. URL: https://arxiv.org/abs/2210.05552.

[Ald85] David J Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. Springer, 1985.

[AMZ06] Sheila Abbas, Mohamed Mosbah, and Akka Zemmari. Distributed computation of a spanning tree in a dynamic graph by mobile agents. In *2006 IEEE International Conference on Engineering of Intelligent Systems*, pages 1–6. IEEE, 2006.

[AOL+14] Gil Ariel, Yotam Ophir, Sagi Levi, Eshel Ben-Jacob, and Amir Ayali. Individual pause-and-go motion is instrumental to the formation and maintenance of swarms of marching locust nymphs. *PloS one*, 9(7):e101636, 2014.

[AP06] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[APB18] Yaniv Altshuler, Alex Pentland, and Alfred M Bruckstein. Introduction to swarm search. In *Swarms and Network Intelligence in Search*, pages 1–14. Springer, 2018.

[APP12] Yaniv Altshuler, Wei Pan, and Alex Sandy Pentland. Trends prediction using social diffusion models. In *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*, pages 97–104. Springer, 2012.

[ASS16] Maksat Atagoziyev, Klaus W Schmidt, and Ece G Schmidt. Lane change scheduling for autonomous vehicles. *IFAC-PapersOnLine*, 49(3):61–66, 2016.

[AYWB08] Yaniv Altshuler, Vladimir Yanovsky, Israel A Wagner, and Alfred M Bruckstein. Efficient cooperative search of smart targets using uav swarms1. *Robotica*, 26(4):551–557, 2008.

[BBH+08] Sepideh Bazazi, Jerome Buhl, Joseph J Hale, Michael L Anstey, Gregory A Sword, Stephen J Simpson, and Iain D Couzin. Collective motion and cannibalism in locust migratory bands. *Current biology*, 18(10):735–739, 2008.

[BC08] Hans-Jurgen Bandelt and Victor Chepoi. Metric graph theory and geometry: a survey. *Contemporary Mathematics*, 453:49–86, 2008.

[BDS08] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 125–140. Springer, 2008.

[BDS13] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 228–243. Springer, 2013.

[BDT13] Zohir Bouzid, Shantanu Das, and Sébastien Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 337–346. IEEE, 2013.

[BM86] Hans-Jürgen Bandelt and Henry Martyn Mulder. Pseudo-modular graphs. *Discrete mathematics*, 62(3):245–260, 1986.

[BMB17] Ariel Barel, Rotem Manor, and Alfred M Bruckstein. Come together: multi-agent geometric consensus. *arXiv preprint arXiv:1902.01455*, 2017.

[BMW97] Alfred M. Bruckstein, C. L. Mallows, and Israel Wagner. Probabilistic pursuits on the grid. *The American Mathematical Monthly*, 104(4):323–343, 1997.

[BN11] Anthony Bonato and Richard J. Nowakowski. *The game of cops and robbers on graphs*. American Mathematical Society, 2011.

[Bro89] Andrei Broder. Generating random spanning trees. In *30th Annual symposium on foundations of computer science*, pages 442–447. IEEE, 1989.

[Bru93] Alfred M. Bruckstein. Why the ant trails look so straight and nice. *The Mathematical Intelligencer*, 15(2):59–62, 1993.

[BS07] Maxim A Batalin and Gaurav S Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675, 2007.

[BSC+06] Jerome Buhl, David JT Sumpter, Iain D Couzin, Joe J Hale, Emma Despland, Edgar R Miller, and Steve J Simpson. From disorder to order in marching locusts. *Science*, 312(5778):1402–1406, 2006.

[BV12] Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702. IEEE, 2012.

[CB17] Anjan Kumar Chandra and Abhik Basu. Diffusion controlled model of opinion dynamics. *Reports in Advances of Physical Sciences*, 1(01):1740008, 2017.

[CD08] Ke Cheng and Prithviraj Dasgupta. Coalition game-based distributed coverage of unknown environments by robot swarms. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1191–1194. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[CGG+08] Hugues Chaté, Francesco Ginelli, Guillaume Grégoire, Fernando Peruani, and Franck Raynaud. Modeling collective motion: variations on the vicsek model. *The European Physical Journal B*, 64(3):451–456, 2008.

[Cha12] Bernard Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55(12):101–110, 2012.

[Cha18] Bernard Chazelle. Toward a theory of markov influence systems and their renormalization. *arXiv preprint arXiv:1802.01208*, 2018.

[CMZ11] T Chou, K Mallick, and RKP Zia. Non-equilibrium statistical mechanics: from a paradigmatic model to biological transport. *Reports on progress in physics*, 74(11):116601, 2011.

[Cor08] Jorge Cortés. Distributed algorithms for reaching consensus on general functions. *Automatica*, 44(3):726–737, 2008.

[CSS00] Debashish Chowdhury, Ludger Santen, and Andreas Schadschneider. Statistical physics of vehicular traffic and some related systems. *Physics Reports*, 329(4-6):199–329, 2000.

[CT94] Wonshik Chee and Masayoshi Tomizuka. Vehicle lane change maneuver in automated highway systems, 1994.

[CVV99] András Czirók, Mária Vicsek, and Tamás Vicsek. Collective motion of organisms in three dimensions. *Physica A: Statistical Mechanics and its Applications*, 264(1-2):299–304, 1999.

[DFGT11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. *Distributed Computing*, 24(3-4):137–147, 2011.

[DGK63] Ludwig Danzer, Branko Grünbaum, and Victor Klee. Helly's theorem and its relatives. *Convexity*, 7:101–180, 1963.

[DHM+09] Erik D Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, Amin S Sayedi-Roshkhar, Shayan Oveisgharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms (TALG)*, 5(3):30, 2009.

[DHM09] Erik D Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. Minimizing movement: fixed-parameter tractability. In *European Symposium on Algorithms*, pages 718–729. Springer, 2009.

[Die17] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, June 2017. URL: https://www.xarg.org/ref/a/3662536218/.

[Dir61] Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1-2):71–76, April 1961.

[DP12] Dariusz Dereniowski and Andrzej Pelc. Drawing maps with advice. *Journal of Parallel and Distributed Computing*, 72(2):132–143, 2012. (Journal version of the same paper from DISC2010).

[DPV15] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44(3):844–867, 2015.

[DZK+18] Yucheng Dong, Min Zhan, Gang Kou, Zhaogang Ding, and Haiming Liang. A survey on the fusion process in opinion dynamics. *Information Fusion*, 43:57–65, 2018.

[EB13] Yotam Elor and Alfred M Bruckstein. *Mathematical Analysis of Emergent Behavior in Multi-Agent Systems*. PhD thesis, Computer Science Department, Technion, 2013.

[EET93] Hossam ElGindy, Hazel Everett, and Godfried Toussaint. Slicing an ear using prune-and-search. *Pattern Recognition Letters*, 14(9):719–722, September 1993. URL: https://doi.org/10.1016/0167-8655(93)90141-y.

[Eke19] Emelie Ekenstedt. Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems, 2019.

[Eps12] Richard A Epstein. *The theory of gambling and statistical logic*. Academic Press, 2012.

[Fey85] Richard Feynman. *"Surely you're joking, Mr. Feynman!": Adventures of a Curious Character*. W.W. Norton, New York, 1985.

[FK10] Natalie Fridman and Gal A. Kaminka. Modeling pedestrian crowd behavior based on a cognitive model of social comparison theory. *Computational and Mathematical Organization Theory*, 16(4):348–372, November 2010. URL: https://doi.org/10.1007/s10588-010-9082-2.

[FS11] Zachary Friggstad and Mohammad R Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms (TALG)*, 7(3):28, 2011.

[FYO+15] Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

[GC13] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[GCJT13] Simon Garnier, Maud Combe, Christian Jost, and Guy Theraulaz. Do ants need to estimate the geometrical properties of trail bifurcations to find an efficient route? a swarm robotics test bed. *PLoS Computational Biology*, 9(3):e1002903, 2013. Dario Floreano, editor. URL: https://doi.org/10.1371%2Fjournal.pcbi.1002903.

[Gib85] Alan Gibbons. *Algorithmic graph theory*. Cambridge university press, 1985.

[GK07] Kevin R Gue and Byung Soo Kim. Puzzle-based storage systems. *Naval Research Logistics (NRL)*, 54(5):556–567, 2007.

[Gol04a] Martin C. Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[Gol04b] Martin C. Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[GR01] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of mathematics and artificial intelligence*, 31(1-4):77–98, 2001.

[GS12] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012.

[HAB+03] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sandor Fekete, and Joseph S. B. Mitchell. Online dispersion algorithms for swarms of robots. In *Proceedings of the nineteenth conference on Computational geometry - SCG 03*. ACM Press, 2003. URL: https://doi.org/10.1145/777792.777854.

[HAB+04] Tien-Ruey Hsiang, Esther M Arkin, Michael A Bender, Sándor P Fekete, and Joseph SB Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V*, pages 77–93. Springer, 2004.

[HAK18] Erez Hartuv, Noa Agmon, and Sarit Kraus. Scheduling spare drones for persistent task performance under energy constraints. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 532–540. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[HB16] Attila Hideg and László Blázovics. Area coverage using distributed randomized methods. In *Cybernetics & Informatics (K&I), 2016*, pages 1–5. IEEE, 2016.

[HIK16] Richard H. Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of Product Graphs, Second Edition (Discrete Mathematics and Its Applications)*. CRC Press, 2016.

[HL17a] Attila Hideg and Tamás Lukovszki. Uniform dispersal of robots with minimum visibility range. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 155–167. Springer, 2017.

[HL17b] Attila Hideg and Tamás Lukovszki. Uniform dispersal of robots with minimum visibility range. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 155–167. Springer, 2017.

[HL20] Attila Hideg and Tamás Lukovszki. Asynchronous filling by myopic luminous robots. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 108–123. Springer, 2020.

[HMS02] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[HMW16] Peter Hegarty, Anders Martinsson, and Edvin Wedin. The hegselmann-krause dynamics on the circle converge. *Journal of Difference Equations and Applications*, 22(11):1720–1731, 2016.

[HOU95] Cem Hatipoglu, Umit Ozguner, and Konur A Unyelioglu. On optimal design of a lane change controller. In *Proceedings of the Intelligent Vehicles' 95. Symposium*, pages 436–441. IEEE, 1995.

[IK00] Wilfried Imrich and Sandi Klavžar. *Product Graphs: Structure and Recognition*. Wiley-Interscience, 2000.

[IK08] Volkan Isler and Nikhil Karnad. The role of information in the cop-robber game. *Theoretical Computer Science*, 399(3):179–190, 2008.

[JCMP17] Stefan Jorgensen, Robert H Chen, Mark B Milam, and Marco Pavone. The risk-sensitive coverage problem: multi-robot routing under uncertainty with service level and survival constraints. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 925–932. IEEE, 2017.

[Joh00] Kurt Johansson. Shape fluctuations and random matrices. *Communications in Mathematical Physics*, 209(2):437–476, February 2000. URL: https://doi.org/10.1007/s002200050027.

[JS+79] Wm Woolsey Johnson, William Edward Story, et al. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879.

[KA19] Ajay D. Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, ICDCN '19, pages 218–227, Bangalore, India. ACM, 2019. URL: http://doi.acm.org/10.1145/3288599.3288610.

[KAGA19] D Knebel, A Ayali, M Guershon, and G Ariel. Intra-versus intergroup variance in collective behavior. *Science advances*, 5(1):eaav0695, 2019.

[KES01] James F. Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm intelligence*. The Morgan Kaufmann series in evolutionary computation. Morgan Kaufmann Publishers, San Francisco, 2001.

[KHM+15] Dominik Krupke, Michael Hemmer, James McLurkin, Yu Zhou, and Sándor P Fekete. A parallel distributed strategy for arraying a scattered robot swarm. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2795–2802. IEEE, 2015.

[KK10] Thomas Kriecherbauer and Joachim Krug. A pedestrian's view on interacting particle systems, kpz universality and random matrices. *Journal of Physics A: Mathematical and Theoretical*, 43(40):403001, 2010.

[Kra12] Eugene F. Krause. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry (Dover Books on Mathematics)*. Dover Publications, 2012.

[KSA+21] Daniel Knebel, Ciona Sha-Ked, Noa Agmon, Gil Ariel, and Amir Ayali. Collective motion as a distinct behavioral state of the individual. *Iscience*, 24(4):102299, 2021.

[KV97] Jakob Krarup and Steven Vajda. On torricelli's geometrical solution to a problem of fermat. *IMA Journal of Management Mathematics*, 8(3):215–224, 1997.

[Law10] Gregory F Lawler. *Random walk and the heat equation*, volume 55. American Mathematical Soc., 2010.

[Lin02] Torgny Lindvall. *Lectures on the coupling method*. Courier Corporation, 2002.

[LL12] Joseph La Salle and Solomon Lefschetz. *Stability by Liapunov's Direct Method with Applications*. Elsevier, 2012.

[LPW09] David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2009.

[LV10] Yaroslav Litus and Richard T Vaughan. Fall in! sorting a group of robots with a continuous controller. In *2010 Canadian Conference on Computer and Robot Vision*, pages 269–276. IEEE, 2010.

[LWZ+15] Zhuofan Liao, Jianxin Wang, Shigeng Zhang, Jiannong Cao, and Geyong Min. Minimizing movement for target coverage and network connectivity in mobile sensor networks. *network*, 4:8, 2015.

[Mat02] J. Matoušek. *Lectures on Discrete Geometry (Graduate Texts in Mathematics)*. Springer, 2002.

[MB18] Rotem Manor and Alfred M Bruckstein. Chase your farthest neighbour. In *Distributed Autonomous Robotic Systems*, pages 103–116. Springer, 2018.

[MG07] Ryan Morlok and Maria Gini. Dispersing robots in an unknown environment. In *Distributed Autonomous Robotic Systems 6*, pages 253–262. Springer, 2007.

[MT93] Sean P. Meyn and Richard L. Tweedie. *Markov Chains and Stochastic Stability*. Springer London, 1993. URL: https://doi.org/10.1007/978-1-4471-3267-7.

[NGGD08] Jose E Naranjo, Carlos Gonzalez, Ricardo Garcia, and Teresa De Pedro. Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver. *IEEE Transactions on Intelligent Transportation Systems*, 9(3):438–450, 2008.

[OFM07] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[PDE+01] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2001.

[Pel05] David Peleg. Distributed coordination algorithms for mobile robot swarms: new directions and challenges. In *International Workshop on Distributed Computing*, pages 1–12. Springer, 2005.

[PSS18] Thomas Petig, Elad M Schiller, and Jukka Suomela. Changing lanes on a highway. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[PSZ09] M. Plumettaz, D. Schindl, and N. Zufferey. Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, 61(5):819–826, April 2009. URL: https://doi.org/10.1057/jors.2009.27.

[RAB21] Dmitry Rabinovich, Michael Amir, and Alfred M. Bruckstein. Optimal physical sorting of mobile agents, 2021. arXiv: 2111.06284.

[RAB22] Ori Rappel, Michael Amir, and Alfred M. Bruckstein. Stigmergy-based, dual-layer coverage of unknown indoor regions, 2022. arXiv: 2209.08573 [cs.MA].

[RMB19] Katja Ried, Thomas Müller, and Hans J Briegel. Modelling collective motion based on the principle of agency: general framework and the case of marching locusts. *PloS one*, 14(2):e0212044, 2019.

[Ros81] Hermann Rost. Non-equilibrium behaviour of a many particle process: density profile and local equilibria. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 58(1):41–53, 1981.

[RW90] Daniel Ratner and Manfred Warmuth. The (n2- 1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

[SA15] Masashi Shiraishi and Yoji Aizawa. Collective patterns of swarm dynamics and the lyapunov analysis of individual behaviors. *Journal of the Physical Society of Japan*, 84(5):054002, 2015.

[SBT+17] Martin Saska, Tomas Baca, Justin Thomas, Jan Chudoba, Libor Preucil, Tomas Krajnik, Jan Faigl, Giuseppe Loianno, and Vijay Kumar. System for deployment of groups of unmanned micro aerial vehicles in gps-denied environments using onboard visual relative localization. *Autonomous Robots*, 41(4):919–944, 2017.

[SCZ14] Beining Shang, Richard Crowder, and Klaus-Peter Zauner. Swarm behavioral sorting based on robotic hardware variation. In *2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*, pages 631–636. IEEE, 2014.

[SCZ16] Beining Shang, Richard M Crowder, and Klaus-Peter Zauner. An approach to sorting swarm robots to optimize performance. In *International Design Engineering Technical Conferences and Computers and Informa-*

tion in Engineering Conference, volume 50152, V05AT07A046. American Society of Mechanical Engineers, 2016.

[SH06] Cheng Shao and Dimitrios Hristu-Varsakelis. Cooperative optimal control: broadening the reach of bio-inspiration. *Bioinspiration & Biomimetics*, 1(1):1–11, March 2006.

[Spi91] Frank Spitzer. Interaction of markov processes. In *Random Walks, Brownian Motion, and Interacting Particle Systems*, pages 66–110. Springer, 1991.

[SSW10] Robin Schubert, Karsten Schulze, and Gerd Wanielik. Situation assessment for automatic lane-change maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):607–616, 2010.

[Szt03] Marcelo Oscar Sztainberg. *Algorithms for Swarm Robotics*. PhD thesis, State University of New York at Stony Brook, 2003.

[TW09] Craig A Tracy and Harold Widom. Asymptotics in asep with step initial condition. *Communications in Mathematical Physics*, 290(1):129–154, 2009.

[VAT+16] F Visintainer, L Altomare, A Toffetti, A Kovacs, and A Amditis. Towards manoeuver negotiation: autonet2030 project from a car maker perspective. *Transportation Research Procedia*, 14:2237–2244, 2016.

[WAYB08] Israel A Wagner, Yaniv Altshuler, Vladimir Yanovski, and Alfred M Bruckstein. Cooperative cleaners: a study in ant robotics. *The International Journal of Robotics Research*, 27(1):127–151, 2008.

[Win07] Peter Winkler. *Mathematical mind-benders*. CRC Press, 2007.

[WLB00] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. Mac versus pc: determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *the International Journal of robotics Research*, 19(1):12–31, 2000.

[WLB97] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. Online graph searching by a smell-oriented vertex process. In *Proceedings of the AAAI Workshop on On-Line Search*. Citeseer, 1997.

[WN06] Alan FT Winfield and Julien Nembrini. Safety in numbers: fault tolerance in robot swarms. *International Journal on Modelling Identification and Control*, 1(ARTICLE):30–37, 2006.

[XWX11] Haoxiang Xia, Huili Wang, and Zhaoguo Xuan. Opinion dynamics: a multidisciplinary review and perspective on future research. *International Journal of Knowledge and Systems Science (IJKSS)*, 2(4):72–91, 2011.

[YAK13] Roi Yehoshua, Noa Agmon, and Gal A Kaminka. Robotic adversarial coverage: introduction and preliminary results. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6000–6005. IEEE, 2013.

[YAK15] Roi Yehoshua, Noa Agmon, and Gal A Kaminka. Frontier-based rtdp: a new approach to solving the robotic adversarial coverage problem. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 861–869. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[YEE+09] Christian A Yates, Radek Erban, Carlos Escudero, Iain D Couzin, Jerome Buhl, Ioannis G Kevrekidis, Philip K Maini, and David JT Sumpter. Inherent noise can facilitate coherence in collective swarm motion. *Proceedings of the National Academy of Sciences*, 106(14):5464–5469, 2009.

[YOA+13] Ercan Yildiz, Asuman Ozdaglar, Daron Acemoglu, Amin Saberi, and Anna Scaglione. Binary opinion dynamics with stubborn agents. *ACM Transactions on Economics and Computation (TEAC)*, 1(4):1–30, 2013.

[ZGM16] Yu Zhou, Ron Goldman, and James McLurkin. An asymmetric distributed method for sorting a robot swarm. *IEEE Robotics and Automation Letters*, 2(1):261–268, 2016.

הניתן לייצוג על-ידי גרף, כאשר הצמתים של הגרף מייצגים מקומות במרחב והקשתות ביניהם מייצגות את היכולת של סוכן לזוז ממקום למקום בצעד זמן אחד. מנקודת מבט תיאורטית, מדובר בתחום שהאנליזה המתמטית שלו נוטה להיות "אד הוק" - כמעט לא ידועים כלים מתמטיים היכולים לשמש אותנו בחקר מערכות מרובות סוכנים על גרפים. אחת מהמטרות של עבודה זו היא לקבץ במקום אחד מספר כלים כאלה במטרה לבנות "ארגז כלים" אשר יכול להוות נקודת התחלה לחקר מערכות מסוג זה. ארגז הכלים שלנו כולל, למשל, את הרעיון של "חילוף זהויות" בין סוכנים. טכניקה זו נובעת מההבחנה שלעיתים ניתן לקחת זוג סוכנים ולהחליף את השמות שלהם על-מנת ליצור סוכנים וירטואליים שקל יותר לנתח אותם מבחינה מתמטית, למשל משום שאופן התזוזה שלהם, או סדר הפעולות שלהם יותר פשוט להבנה. טכניקות אחרות שנדבר עליהן כוללות "פונקציות פוטנציאל", צימוד (טכניקה בתורת ההסתברות), מערכות "חלקיקים מתקשרים", והתפלגויות סטציונאריות של שרשראות מרקוב.

תקציר

מערכות מרובות סוכנים הן תחום אינטרדיסציפלינרי רחב, שיש לו ישומים ברובוטיקה, מערכות מבוזרות, תורת המשחקים, ביולוגיה, וסוציולוגיה. מערכת מרובת סוכנים היא מערכת מבוזרת המורכבת ממספר סוכנים אוטונומיים אשר משתפים פעולה בכדי להשיג מטרה מסוימת. עבודה זו עוסקת באופן שבו אינטראקציות לוקאליות בין סוכנים פשוטים יכולות להוביל למצב גלובאלי רצוי. נחקור נושא זה משתי פרספקטיבות: פרספקטיבה של ביולוג הצופה בעולם הטבע, ופרספקטיבה של מהנדס המבקש לבנות אלגוריתמים שניתן ליישמם בתחום הרובוטיקה הנחילית. בעולם הטבע, פעמים רבות אורגניזמים משתפים פעולה באופן ספונטני: נמלים יוצרות או מוצאות שבילים ישרים, נחילים של מיליוני חגבים מתכנסים לכיוון תעופה זהה, גחליליות מתאמות הבזקי אור כאילו היה בידיהן שעון חול. בחלק הראשון של עבודה זו, העוסק בפרספקטיבה הביולוגית, המטרה שלנו תהיה להבין לעומק את התהליכים הללו ולמדל את כללי ההתנהגות העומדים מאחוריהם ברמת הסוכן, כשהשאלה שמעניינת אותנו היא מה ניתן ללמוד מהאלגוריתמים של "אמא טבע". לעומת זאת, בחלק השני של עבודה זו נהפוך למהנדסים, ונשאל כיצד נוכל ליצור כללים סוכניים המובילים לביצועים נכונים ויעילים של נחילי רובוטים. במהלך עבודה זו נראה שבאמצעות האלגוריתם הלוקאלי הנכון, רובוטים נחיליים יכולים למפות סביבות בלתי ידועות, לתקן שגיאות ומצבי אי-סיפוק הנוצרים מהתרסקויות של חלק מהנחיל, לזהות ולחלק מטלות באופן שווה, וליעל תהליכי תנועה של רכבים אוטונומיים.

בכוונתנו למדל נחילי רובוטים ונחילים של אורגניזמים טבעיים תחת אותה פרדיגמה, וכך לאפשר גם השוואה וגם שימוש ב- ופיתוח של טכניקות משותפות לשני הנושאים. מאחורי גישה זו עומד תחום הקרוי "אלגוריתמים טבעיים", אשר שואף להשתמש בכלים מרובוטיקה וממדעי המחשב בכדי להבין תופעות טבעיות כגון שבילי נמלים או מעוף ציפורים, כמו גם להשתמש בתופעות מעולם הטבע כמקור השראה לאלגוריתמים, וזאת באמצעות תרגום של תופעות אלה למודלים אשר נעשה בהם שימוש בתחום הרובוטיקה הנחילית. הפרדיגמה שבחרנו להתרכז בה היא פרדיגמת "הרובוטיקה הנמלית", אשר מניחה שהרובוטים ששייכים לנחיל מוגבלים מאד מבחינת טווח החישה שלהם, הכוח החישובי שלהם, היכולת שלהם לזכור אינפורמציה מנקודות זמן קודמות, והיכולת שלהם לתקשר אחד עם השני. מקור ההשראה של פרדיגמת הרובוטיקה הנמלית היא היכולת של נחילים של חיות חברתיות כגון ציפורים, נמלים או חגבים לשתף פעולה ולהשיג תוצאות גלובאליות רצויות על אף יכולות החישה, התכנון, והחישוב המוגבלות של כל חיה ברמה האינדיבידואלית. אנחנו משתמשים בפרדיגמה זו גם בבואנו לתכנן אלגוריתמים עבור רובוטים נחיליים עם מטרה מוגדרת מראש, וגם ככלל מנחה בבואנו למדל התנהגות נחילית של אורגניזמים טבעיים באופן פורמלי, כאילו היו מונחים על-ידי אלגוריתם רובוטי.

כמעט כל המודלים המתמטיים שנעסוק בהם בעבודה זו מניחות שהסוכנים זזים בחלל סופי ודיסקרטי

המחקר בוצע בהנחייתו של פרופסור אלפרד ברוקשטיין, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקר הדוקטורט של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Michael Amir and Alfred M Bruckstein. Probabilistic pursuits on graphs. *Theoretical Computer Science*, 2019.

Michael Amir and Alfred M. Bruckstein. Minimizing Travel in the Uniform Dispersal Problem for Robotic Sensors. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 113–121, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems, 2019. (Visited on 07/30/2022).

Michael Amir and Alfred M. Bruckstein. Fast uniform dispersion of a crash-prone swarm. In *Proceedings of Robotics: Science and Systems*, RSS '20, 2020.

Michael Amir, Noa Agmon, and Alfred M Bruckstein. A discrete model of collective marching on rings. *International Symposium Distributed Autonomous Robotic Systems*:320–334, 2021.

Dmitry Rabinovich*, Michael Amir*, and Alfred M. Bruckstein. Optimal physical sorting of mobile agents, 2021. arXiv: 2111.06284.

Michael Amir, Noa Agmon, and Alfred M. Bruckstein. A locust-inspired model of collective marching on rings. *Entropy*, 24(7):918, June 2022.

Ori Rappel*, Michael Amir*, and Alfred M. Bruckstein. Stigmergy-based, dual-layer coverage of unknown indoor regions, 2022. arXiv: 2209.08573 [cs.MA].

Michael Amir, Yigal Koifman, Yakov Bloch, Ariel Barel, and Alfred M. Bruckstein. Multi-agent distributed and decentralized geometric task allocation, 2022. arXiv: 2210.05552 [cs.MA].

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

מערכות מרובות סוכנים על גרפים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

מיכאל אמיר

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
תשרי התשפג חיפה אוקטובר 2022

מערכות מרובות סוכנים על גרפים

מיכאל אמיר