The Dissertation Committee for Kathryn Long Genter

certifies that this is the approved version of the following dissertation:

Fly with Me: Algorithms and Methods for Influencing a Flock

Committee:

Peter Stone, Supervisor

Noa Agmon

Risto Miikkulainen

Bruce Porter

Fly with Me: Algorithms and Methods for Influencing a Flock

by

Kathryn Long Genter

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2017

To my parents, Gene and Glenda Long, and my husband, JT Genter,
for their unconditional love, support, and encouragement.

Acknowledgments

My first experience with Peter Stone and his research group was at a RoboCup competition. I was volunteering at RoboCup 2007 in Atlanta and had started following the progress of Peter's @Home team after serving as a test subject in the "Follow Me" task. Little did I know at RoboCup 2007 that Peter would eventually become my PhD advisor. Peter takes his commitment to his PhD students seriously. Among other things, Peter is responsive, open to discussion and debate, and helpful at brainstorming. I've enjoyed working with Peter and I've appreciated his guidance during my PhD.

Peter tends to attract great researchers — that also tend to be wonderful people — to his group. My PhD would certainly not have been the same with a different group. I'd like to thank all of the LARG group, past and present, for their support, conversations, and friendship. Special thanks go to Patrick MacAlpine for providing especially useful suggestions at multiple points during my PhD.

I'm thankful for my entire committee. As a post-doc in Peter's group, Noa Agmon significantly helped me with my early theoretical thesis work as well as work on another project. Her strong theoretical abilities and her willingness to mentor and teach me are much appreciated. Bruce Porter and Risto Miikkulainen both provided valuable time and input on multiple occasions which I appreciate as well.

I'm also appreciative of my mentors during my undergraduate degree at Georgia Tech. Monica Sweat provided useful advice and guidance throughout my undergraduate degree. Zsolt Kira mentored me as part of the SAIC Scholars program for multiple years before I transitioned to working with Ashwin Ram and his research group on my undergraduate research thesis. Monica, Zsolt, and Ashwin all played a large role in preparing me for the research described in this dissertation.

I'd like to thank the UT Austin Villa RoboCup Standard Platform League (SPL) team.

I've enjoyed working in our codebase over the years and realize that the team would not have been nearly as successful without the contributions from many teammates. I'd also like to thank everyone who served with me on the SPL technical committees and organizing committees over the years. I'd especially like to thank Tim Laue for his partnership in organizing, planning, running, and writing about the SPL drop-in player competition across multiple years.

The UT RecSports Outdoor Adventure Trip program played a large part in the second half of my PhD. Participating in guide school in Fall 2013 and leading many subsequent trips have both been incredibly healthy for me.

A big thank you to my bi-weekly racquetball group for giving me a consistent competitive outlet throughout most of my PhD. I'd also like to thank the UT Club Racquetball team for five years of competitive racquetball and four trips to the intercollegiate racquetball national championships.

I'd like to thank my parents, Gene and Glenda Long, for always believing in me and supporting me. My parents advocated for me when I was young and ensured that I had the help, support, and environment necessary to succeed. My childhood tutor, Jane Smith, worked with me multiple days a week from 1st grade to 6th grade to improve my reading, writing, and spelling skills — without her teaching in my early years, this dissertation would not have been possible. I'd like to thank my grandfather, Ken Spiess, for encouraging me to live life to the fullest. Finally, many thanks to my husband JT Genter, who has been with me through my PhD. From moving to Texas with me to sticking with me through rough times to financially supporting us through my graduate school — he's been a constant throughout that I sincerely and greatly appreciate.

Kathryn Long Genter

The University of Texas at Austin

August 2017

Fly with Me: Algorithms and Methods for Influencing a Flock

Publication No. _____

Kathryn Long Genter, Ph.D.
The University of Texas at Austin, 2017

Supervisor: Peter Stone

As robots become more affordable, they will begin to exist in the world in greater quantities. Some of these robots will likely be designed to act as components in specific teams. These teams could work on tasks that are too large or complex for a single robot — or that are merely more efficiently accomplished by a team — such as surveillance in a large building or product delivery to packers in a warehouse. Multiagent systems research studies how these teams are formed and how they work together.

Ad hoc teamwork, a newer area of multiagent systems research, studies how new robots can join these pre-existing teams and assist the team in accomplishing its goal. This dissertation extends and applies research in ad hoc teamwork towards the general area of flocking, which is an emergent swarm behavior. In particular, the work in this dissertation considers how ad hoc agents — called influencing agents in this dissertation — can join a flock, be recognized by the rest of the flock as part of the flock, influence the flock towards particular behaviors through their own behavior, and then separate from the flock. Specifically, the primary research question addressed in this dissertation is **How can influencing agents be utilized in various types of flocks to**

influence the flock towards a particular behavior?

In order to address this research question, this dissertation makes six main types of contributions. First, this dissertation formalizes the problem of using influencing agents to influence a flock. Second, this dissertation contributes and analyzes algorithms for influencing a flock to a desired orientation. Third, this dissertation presents methods for determining how to best add influencing agents to a flock. Fourth, this dissertation provides methods by which influencing agents can join and then leave a flock in motion. Fifth, this dissertation evaluates some of the influencing agent algorithms on a robot platform. Sixth, although the majority of this dissertation assumes the influencing agents will join a flock that behaves similarly to European starlings, this dissertation also provides insight into when and how its algorithms are generalizable to other types of flocks as well as to general teamwork and coordination research. All of the methods presented in this dissertation are empirically evaluated using a simulator that can support large flocks.

Contents

List of Tables

List of Figures

1. Introduction

Birdstrikes to aircraft cost the United States aviation industry over $625 million in damages annually and have resulted in at least 200 human deaths since 1990 [62]. A birdstrike is a collision between an airborne animal and a human-made structure — most commonly a bird with an aircraft. However, birdstrikes also occur between animals and wind farms, as well as animals and vehicles and animals and buildings. According to the *Managing Raptors at Airports* talk at the 2014 Bird Strike Committee US Meeting,[1] the most common methods used to decrease birdstrikes include habitat management to reduce prey abundance, non-lethal hazing (scaring the birds away), live-capture and relocation, and lethal control (actively killing the birds). All of these methods are highly invasive for the animal — some even result in the animal being purposefully killed — as well as costly to the organizations carrying out these procedures.

Deadly birdstrikes usually involve a large group of animals — often birds — coming into contact with an aircraft. Groups of agents — living, simulated, or robot — are said to be "swarming" when they move together in a formation or cluster. When birds form a swarm, they are considered to be a flock. Flocking is an emergent swarm behavior in which each bird in the flock follows a simple local behavior rule under which its behavior is determined by the behavior of nearby birds. When all of the birds in a flock follow this simple behavior rule, the resulting flock behavior appears well organized.

To the best of our knowledge, there is no way to directly and predictably control the flight path of birds. Various current methods, such as non-lethal hazing, keep birds away from particular areas but are unable to control where the birds fly instead. The work in this dissertation considers how to influence a flock towards a particular behavior — such as avoiding airports — by adding *influencing agents* to a flock composed of *flocking agents*. These influencing agents — which could be in the form of robot birds or ultralight aircraft — then attempt to influence the flocking agents merely by being perceived by the rest of the flock as another member of the flock.

[1] `http://events.aaae.org/sites/140804/index.cfm`

Although on the surface it may seem quite futuristic, it is not infeasible that influencing agents in the form of robot birds could be used to influence flocks of birds in real life. The makers of the Robird[2] robot bird mentioned at the 2015 North American Bird Strike Conference that in making their predator Robird, they first accidentally created prototypes that bird species in the Netherlands attempted to flock with during test flights. In Operation Migration,[3] captive-hatched whooping cranes were trained to imprint on humans wearing whooping crane costumes. The young whooping cranes then followed a costumed human as he flew an ultralight plane along a migration route. Recently, scientists used a microlight plane to show hand-raised northern bald ibises their ancestral migration route in Europe [67].

Birdstrikes in various forms continue to be a threat. This dissertation is motivated in large part by the hypothesis that influencing agents could provide a viable, preferable alternative to reducing birdstrikes. With this motivation in mind, this dissertation presents algorithms that influencing agents could use in order to influence swarms to avoid dangerous and/or undesirable areas such as airports, wind farms, city streets, and agricultural areas while swarming. Although the algorithms presented in this dissertation are designed to influence flocks that behave similarly to European starlings, we also discuss how our algorithms and ideas could be applicable to other types of flocks and teams.

The work described in this dissertation falls within two main research areas: swarming and multiagent systems. In the following sections, we introduce both of these areas and discuss specifically how using influencing agents to alter the behavior of a flock relates to these areas.

1.1 Swarming

A swarm is a large, dense group of animals. Swarms are usually thought to be composed of flying animals — such as bees, locusts, or wasps — but they could also be composed of land-based animals such as sheep, ants, or even humans. Additionally, swarms can be composed of living animals, simulated animals, robot animals, or a mixture of living, simulated, and robot animals. Within swarms, each animal is usually rather naive regarding the overall behavior of the swarm.

[2] http://clearflightsolutions.com/methods/robirds
[3] http://operationmigration.org/

Additionally, no animal is "in charge" or a "leader" in a swarm. Instead, the animals in the swarm self-organize by each individual agent following a simple behavior and simple interaction scheme. The resulting behavior of the entire swarm often ends up being more complex and well organized than any of the individual animals could have orchestrated.

Swarming can be seen in many different variants. Some common variants include birds flocking, quadrupeds herding, and fish schooling. Although the methods presented in this dissertation could be applied to various swarm variants, this dissertation focuses on flocking.

Flocking is an emergent swarm behavior found in various species in nature, but most commonly in birds. Each animal in a flock follows a simple local behavior rule, but this simple behavior by each individual agent often results in group behavior that appears well organized and stable. Following a well-recognized algorithm for flocking [68], in this dissertation we assume that each bird in the flock dynamically adjusts its behavior based on that of its immediate neighbors.

Flocking is often studied under the assumption that all of the agents are identical or represent a small set of well-defined behavior types. Indeed, various disciplines such as physics [80], graphics [68], biology [19], and distributed control theory [42, 50, 75] have studied flocking in order to characterize its emergent behavior. In this dissertation we instead focus on the problem of *leading* a flock to adopt particular behaviors by adding a small number of controllable agents to the flock. In particular, we assume that we are given a flock whose members follow a known, well-defined rule characterizing their behavior and we wish to examine to what extent it is possible to influence the flock.

In this dissertation, we assume that the swarm is designed by other people (in the case of a swarm composed of robot or simulated animals) or by nature (in the case of a swarm composed of living animals). Hence, we assume that we are unable to explicitly alter the swarm. As such, the influencing agents we design must work with the behaviors, sensors, actuators and capabilities that the animals in the swarm currently possess. Specifically, the influencing agents join and influence a pre-existing "flock" (team) whose programs cannot be altered. The agents in the pre-existing team are already following a set flocking behavior. Hence, if we want to modify the team's behavior (such as changing its heading), we can add one or more influencing agents to the team and these influencing agents can influence the team towards the desired behavior by behaving in a particular

manner.

As a motivating example, consider a flock of migrating birds. Assume there is an airport on the flock's desired path between its current location and its desired location. If left alone, the flock will travel over the airport and potentially collide with an aircraft. Such a birdstrike is undesirable because it could kill members of the flock, incur costly damages to the aircraft, cause delays at the airport, and potentially kill passengers on the aircraft. The birds could be chased away from the airport with predators or loud noises, but this is not ideal since these are disturbing to the birds and ineffectively control how the birds will behave instead. Instead, influencing agents — such as those described in this dissertation — could be utilized to encourage the flock to alter its collective flight path as desired. These influencing agents follow specified algorithms but are perceived by the rest of the flock to be one of their own. Hence, the flock still has no explicit leader and does not know that the influencing agents are attempting to influence the flock to behave in a particular way.

1.2 Multiagent Systems

Within the context of the broad field of Artificial Intelligence, this dissertation is most related to the sub-area of multiagent systems. *Multiagent systems* contain multiple intelligent agents interacting within an environment. An *agent* is an autonomous entity that observes its environment and acts upon its environment and an *intelligent agent* is an agent that purposefully acts in order to obtain a set goal. A *multirobot system* is a type of multiagent system in which the intelligent agents are robots.

Multiagent systems are often able to solve problems and accomplish tasks that would be difficult or impossible for a single agent, such as collective box pushing in which multiple agents are needed in order to move a box and patrolling in which multiple agents are able to patrol an area more securely and effectively than just one agent. Unlike centrally-controlled systems, the intelligent agents comprising a multiagent system collectively make all of the decisions for the multiagent system.

Flocks can be seen as a type of multiagent system. The birds comprising the flock can be

considered to be intelligent agents that are all influencing each other towards the flock's overall goal. This overall goal might at any point be hunting, migrating, avoiding a predator, or merely flying towards a tree to rest. The overall flock behavior that results from each agent's behavior usually appears to be surprisingly well-coordinated and effective at accomplishing the flock's goal.

One sub-area of multiagent systems research concerns *teamwork*. Teamwork in multiagent systems is distinguished from other types of multiagent systems research in that all of the agents have the same goals. As discussed in detail in Chapter 9, most previous and ongoing work on teamwork in multiagent systems has dealt with how to design entire teams of intelligent agents to work together to accomplish large tasks or goals. However, in this dissertation we consider how to design one or more intelligent agents to add to a pre-existing team — which falls within the scope of a recently introduced subfield of multiagent systems called *ad hoc teamwork* [71].

Ad hoc teamwork is different from most research on teamwork because it focuses on creating agents that can cooperate with unknown teammates without prior coordination [71]. Agents on an ad hoc team have shared goals, are not able to pre-coordinate with each other, and have no explicit communication with each other. Additionally, an ad hoc team is mainly comprised of agents that are not directly controllable — we are only in control of a small portion of the ad hoc team. Past work in ad hoc teamwork is described in detail in Chapter 9.

In this dissertation, the influencing agents we design are ad hoc teammates that join pre-existing flocks and work as members of the flock to assist in accomplishing the flock's goal.

1.3 This Dissertation

Following the preceding motivation, this dissertation focuses on answering the following question:

> How can influencing agents be utilized in various types of flocks in order to influence these flocks towards a particular behavior?

In order to answer this question, this dissertation provides the following contributions.

- **Chapter 2 — Problem definition**

This dissertation begins by defining the assumptions, parameters, and objectives for the problem of adding influencing agents to a flock.

- **Chapter 3 — Algorithm for leading a stationary flock to a desired orientation**
  This dissertation sets bounds on the extent of influence the influencing agents can have on the team when the agents are stationary. Additionally, this dissertation contributes an algorithm for orienting a stationary flock to a desired orientation using a set of non-stationary influencing agents. This algorithm is analyzed both theoretically and empirically.

- **Chapter 4 — Algorithms for influencing a flock to a desired orientation**
  Directing a flock away from danger requires being able to influence a flock to alter its orientation. As such, this dissertation contributes three algorithms, as well as detailed experimental results for all three algorithms, that can be used by influencing agents to influence a flock to orient towards a desired orientation. This dissertation also experimentally considers how to use at least one of these algorithms to maneuver the flock through turns quickly but with minimal agents being separated from the flock as a result of these turns.

- **Chapter 5 — Methods for placing influencing agents into a flock**
  Influencing agents in different parts of a flock have different influence over the flock. Hence, determining how to place influencing agents into a flock if given the opportunity is important. As such, this dissertation considers various methods for placing influencing agents directly into a flock. Each method is empirically evaluated in this dissertation.

- **Chapter 6 — Methods for influencing agents to join and leave a flock**
  It is not realistic to assume that influencing agents can always be placed directly into a flock — they may instead need to join the flock from somewhere outside the flock, influence the flock, and then eventually leave the flock. However, joining and leaving a flock is not trivial because the influencing agents will influence the flock in unintended and/or unavoidable ways as they join and leave. Hence, this dissertation contributes various methods by which influencing agents could join and leave a flock already in motion while decreasing the negative influence joining and leaving may have on the flock. Specifically, this dissertation addresses the scenario

where a flock is currently flying and influencing agents need to intercept the flock, enter the flock, influence the flock in a particular manner, and then carefully extract themselves from the flock. Each method is empirically evaluated in this dissertation.

- **Chapter 7 — Evaluate influencing agent behavior and placement algorithms on flocks with different behaviors**

  The contributions presented in Chapters 3-6 assume the influencing agents will join a flock that exhibits a particular type of flocking behavior. However, there are many possible variants of flock-member behavior. As such, we evaluate how well the algorithms presented in this dissertation perform when the flock members exhibit a different type of flocking behavior.

- **Chapter 8 — Implementation on a robot platform**

  Taking algorithms from simulation to the real world is often difficult, yet it is an important development step. Implementation on a robot platform often exposes the infeasibility of some underlying algorithm or model assumptions, and thus can help motivate inclusion of more realistic assumptions and direct the future development of algorithms. As such we implemented and evaluated one of the algorithms from Chapter 4 on multiple SoftBank Robotics NAO robots. The experiments are reported in this dissertation. These experiments consider how influencing agents can influence a flock of bipedal robots to avoid a particular area.

Although this dissertation is intended to be read in order from start to finish, it is understandable that some readers may want to survey particular chapters in isolation. For readers who wish to read chapters out of order, Figure 1.1 indicates the relations between the different chapters. For example, if you want to read Chapter 6, then it is important to read Chapters 2 and 4 first and we recommend that you also read Chapter 5. However, it is not important to read Chapter 3 before Chapter 6.

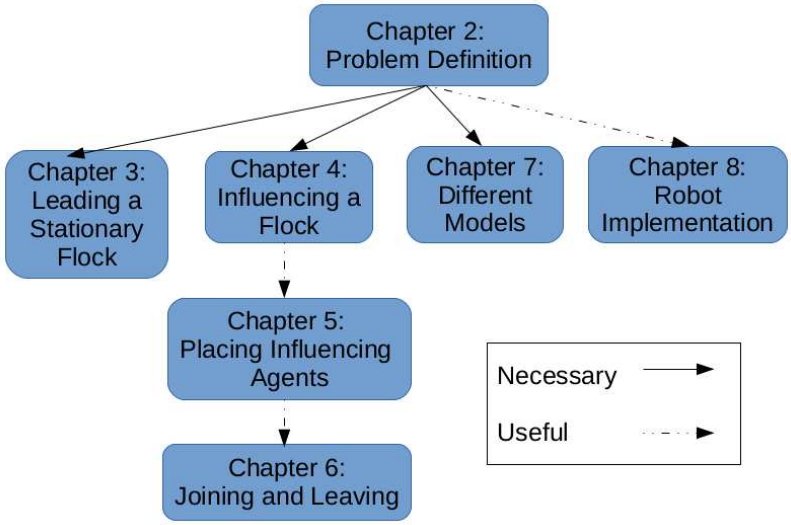


Figure 1.1: Guide for reading individual chapters: a solid arrow is drawn from a chapter that is necessary as a background for a following chapter, and a dashed arrow is drawn from a chapter that is useful as a background for a following chapter.

2. Problem Definition

To fully define our problem, in this chapter we specify (1) our flocking model and (2) the performance objective. At the end of this chapter, we also introduce the simulation environments under which we run our experiments.

2.1 Flocking Model

A flocking model defines how members of a flock behave. For the purposes of this dissertation, a flocking model is composed of (1) a neighborhood model and (2) an influence model.

A neighborhood model defines which agents are able to influence a flocking agent. Two examples of common neighborhood models for flocks are (1) x nearest neighbors and (2) agents within y distance. Sometimes closer neighbors are weighted more heavily than farther neighbors. Section 2.1.1 introduces the neighborhood models utilized in this dissertation.

An influence model defines how the flocking agents update their behavior at each time step. Influence models often consider environmental factors (such as wind, surrounding agents, and predators) as well as internal factors (such as current heading and hunger level). Section 2.1.2 introduces the primary influence model utilized in this dissertation.

2.1.1 Neighborhood Model

Most flocking models accept that flocking agents are only influenced by other agents (both influencing and flocking) that are located within their *neighborhood*. A neighborhood model defines how an agent's neighborhood is calculated.

Throughout most of this dissertation we use a *visibility radius* as our neighborhood model. Most biologists agree that interaction depends on metric distance, although they often disagree on the specifics [6, 13]. As such, we use a simple metric distance method — a *visibility radius* — as our primary neighborhood model. Specifically we let $N_i(t)$ be the set of $n_i(t) \leq n$ agents (not including

agent a_i) at time t which are located within the *visibility radius* r of agent a_i. All agents in $N_i(t)$ are considered to be neighbors of a_i and thus influence a_i. See Figure 2.1 for a pictorial description of the *visibility radius* neighborhood model.

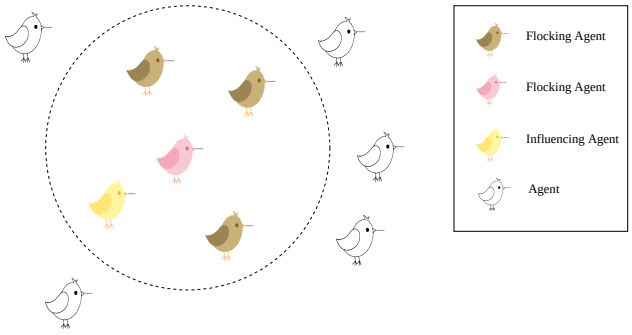


Figure 2.1: A diagram explaining the *visibility radius* neighborhood model. The yellow and brown agents are within the pink agent's neighborhood, while the clear agents are not within the pink agent's neighborhood.

Regardless of the neighborhood model utilized, the number of influencing agents inside agent a_i's neighborhood is denoted by $k_i(t)$ and the number of flocking agents inside agent a_i's neighborhood is denoted by $m_i(t)$, where $k_i(t) + m_i(t) = n_i(t)$.

Alternate Neighborhood Models

Although the *visibility radius* neighborhood model is utilized throughout most of this dissertation, we do utilize multiple other neighborhood models. Specifically, the work presented in Chapter 3 utilizes a *visibility sector* neighborhood model and Chapter 7 explores how the methods and algorithms in this dissertation perform with *visibility sector*, *N-nearest neighbors*, and *weighted influence* neighborhood models. Below, we introduce these three alternate neighborhood models.

For each alternate neighborhood model, remember that the neighborhood model defines which agents are in $N_i(t)$ for agent a_i. Regardless of the neighborhood model, all of the agents within $N_i(t)$ influence a_i.

Chapters 3 and 7 utilize a *visibility sector* neighborhood model. It is generally accepted that birds have a "blind" angle behind them such that any neighboring birds within this "blind" area are not considered when performing orientation updates [45]. This "blind" area is the motivation for our *visibility sector* neighborhood model. For the *visibility sector* neighborhood model we let

$N_i(t)$ be the set of $n_i(t) \leq n$ agents (including agent a_i) at time t which are visible to agent a_i. An agent is *visible* to agent a_i if its position is located within a *visibility sector* of angle α centered on orientation $\theta_i(t)$ (see Figure 2.2 for an example). For ease, Chapter 3 assumes that the *visibility sector* extends from agent a_i for an unlimited distance, while Chapter 7 assumes that the *visibility sector* extends from agent a_i for a finite distance r. We say that angle α — and radius r in Chapter 7 — defines the *visibility sector* for each agent, and that this *visibility sector* defines each agent's *neighborhood* (i.e., the area in which the agent can see other agents).

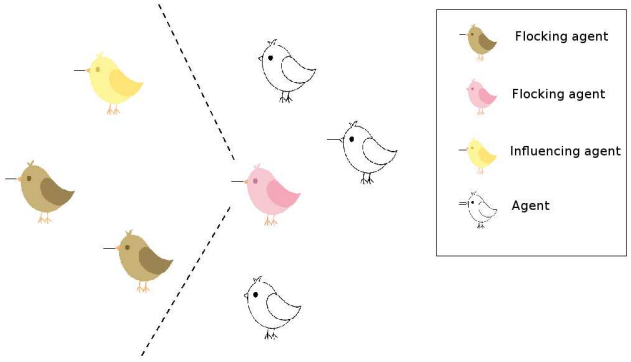


Figure 2.2: A diagram explaining the *visibility sector* neighborhood model. The yellow and brown agents are within the pink agent's neighborhood, while the clear agents are not within the pink agent's neighborhood.

Chapter 7 utilizes two additional neighborhood models: *N-nearest neighbors* and *weighted influence*. The motivation for the *N-nearest neighbors* neighborhood model is that starlings are believed to consider the seven nearest birds in their flock as their neighborhood when performing orientation updates [6, 13]. As such, the *N-nearest neighbors* neighborhood model fills $N_i(t)$ with the N agents located physically nearest to agent a_i at time t. The *weighted influence* neighborhood model is motivated by the idea that animals are most strongly influenced by those physically nearest to them. Hence, the *weighted influence* neighborhood model functions similarly to the *visibility radius* neighborhood model except that it weights each neighbor such that neighbors physically closer to agent a_i have more influence over a_i while agents physically farther from a_i have less influence over a_i. Neighbors are weighted linearly, such that a neighbor x distance from a_i will have half the influence of a neighbor $\frac{x}{2}$ distance from a_i. The decision to weight the neighbors linearly was arbitrary, but — as we discuss in Chapter 7 — results led us to not subsequently

consider other functions. The neighbors are weighted such that the total influence is the same across all neighbors for the *visibility radius* and *weighted influence* neighborhood models.

2.1.2 Influence Model

An influence model defines how each flocking agents updates its behavior at each time step as a function of the agents within its neighborhood. In this subsection, we describe the primary influence model used in this dissertation.

We assume that there are two types of agents: k *influencing agents* and m *flocking agents*. As such, there are $n = k + m$ total agents in the environment. Every agent a_i has a velocity $v_i(t)$, a position $p_i(t)$, and an orientation $\theta_i(t)$ at time t. Note that $v_i(t)$, $p_i(t)$, and $\theta_i(t)$ are dependent on the time t and can be different for each agent a_i. Each agent's position $p_i(t) = (x_i(t), y_i(t))$ at time t is updated after its orientation is updated. Hence, $x_i(t) = x_i(t-1) + v_i(t)\cos(\theta_i(t))$ and $y_i(t) = y_i(t-1) - v_i(t)\sin(\theta_i(t))$.

The k influencing agents $\{a_0, \ldots, a_{k-1}\}$ are controlled algorithmically via the algorithms and behaviors presented in this dissertation while the m flocking agents $\{a_k, \ldots, a_{N-1}\}$ behave according to the influence model. Except where specified, the influence model for the flocking agents in this dissertation is defined by a simplified version of Reynolds' Boid algorithm for flocking [68].

Reynolds' Boid algorithm for flocking is comprised of three aspects: *separation*, *alignment*, and *cohesion*. Vectors representing each aspect are added together to determine the behavior of each agent. The *separation* aspect steers each agent away from its neighbors to avoid collisions. The *alignment* aspect steers each agent towards the average heading of its neighbors. The *cohesion* aspect steers each agent towards the average position of its neighbors. Together, these aspects allow the flocking agents to behave similarly to real-life flocks.

Although Reynolds' algorithm for flocking is comprised of three aspects, the influence model used throughout most of this dissertation utilizes only the *alignment* aspect. This influence model was chosen because it is similar to the models used by Jadbabaie *et al.* [50] and Vicsek *et al.* [80]. In Chapter 7 we show how the algorithms and behaviors from this dissertation perform when the flocking agents behave according to the full Reynolds' algorithm for flocking.

Under the *alignment* influence model utilized throughout most of this dissertation, the flock-

ing agents update their orientations based on the orientations of their neighbors. Hence, the global orientation of agent a_i at time step $t+1$, denoted by $\theta_i(t+1)$, is set to be the average orientation of all agents in $N_i(t)$ at time t. Formally,

$$\theta_i(t+1) = \theta_i(t) + \frac{1}{n_i(t)} \sum_{a_j \in N_i(t)} \text{calcDiff}(\theta_j(t), \theta_i(t)) \tag{2.1}$$

We use Equation 2.1 instead of taking the average orientation of all agents because of the special cases handled by Algorithm 1. For example, the mathematical average of 350° and 10° is 180°, but by Algorithm 1 it is 0°. Throughout this dissertation, we restrict $\theta_i(t)$ to be within $[0, 2\pi)$.

Algorithm 1 calcDiff$(\theta_i(t), \theta_j(t))$

```
1: if ((θ_i(t) − θ_j(t) ≥ −π) ∧ (θ_i(t) − θ_j(t) ≤ π)) then
2:     return θ_i(t) − θ_j(t)
3: else if θ_i(t) − θ_j(t) < −π then
4:     return 2π + (θ_i(t) − θ_j(t))
5: else
6:     return (θ_i(t) − θ_j(t)) − 2π
```

2.2 Performance Representation

Each chapter of this dissertation introduces contributions towards solving different problems. In this section, we describe the performance metrics suited to the different types of problems faced in this dissertation.

2.2.1 Flock Manipulation

During each time step, the influencing agents first orient to their desired orientations based on some plan π. Next, the flocking agents update their orientations based on the orientations of all the agents in their neighborhoods (using Equation 2.1). Finally, the positions of all the agents are updated.

A plan for x time steps for an agent a_i (denoted by π_x^i) is a set of x orientations the agent should execute (one per time step), i.e., $\pi_x^i = (\theta_i(0), \theta_i(1), \ldots, \theta_i(x-1))$. π_x is the set of plans of x steps for all influencing agent, i.e., $\pi_x = (\pi_x^0, \pi_x^1, \ldots, \pi_x^{k-1})$. The *performance error* $E(\pi_x)$ of π_x

is the sum of the differences between each flocking agent's final orientation after x steps and θ^*, formally

$$E(\pi_x) = \sum_{j=k}^{n-1} |\text{calcDiff}(\theta^*, \theta_j(x))| \tag{2.2}$$

The *cost* of π_x is defined as

$$c(\pi_x) = w_1 x + w_2 E(\pi_x) \tag{2.3}$$

where w_1 is a weight that can be set to emphasize the importance of lesser time steps, x is a scalar representing the length of the plan π_x, and w_2 is a weight that can be set to emphasize the importance of lower performance error. At the extremes, setting $w_1 >> w_2$ encourages finding reasonably low performance error in as few steps as possible, while setting $w_2 >> w_1$ encourages minimizing performance error using as many steps as are needed.

An optimal plan π^* is one with minimal cost $c(\pi^*)$. For optimal plan π^*, performance error decreases when more time steps are available such that $E(\pi^*_0) \geq E(\pi^*_1) \geq E(\pi^*_2) \geq \ldots \geq E(\pi^*_\infty)$. Performance error never increases as more time steps are available for an optimal plan because the optimal behavior given one additional time step is to either influence the same as with one fewer time step (and obtain the same performance error) or influence at least one flocking agent to orient itself closer to θ^* (and obtain lower performance error). The optimal number of time steps $|x|$ for a task is the x at which cost $c(\pi_x)$ is minimal. Likewise, the optimal cost $|c(\pi)|$ is equal to $c(\pi_{|x|})$.

In Chapters 3 and 4 of this dissertation, we set w_1 to a small positive value (the exact value does not matter, as long as it is significantly less than ∞) and set w_2 to ∞. With these settings for w_1 and w_2 we obtain the least-step plan in which all flocking agents orient to θ^*, if such a plan exists. If such a plan does not exist, then we obtain a plan with low performance error that uses as few steps as possible.

Chapters 3 and 4 are mainly concerned with the Agent Flock Orientation Manipulation Problem. We define the Agent Flock Orientation Manipulation Problem as follows:

Given a target orientation θ^* and a team of n homogeneous agents $\{a_0, \ldots, a_{n-1}\}$, where the flocking agents $\{a_k, \ldots a_{n-1}\}$ calculate their orientation based on Equation 2.1, determine whether the influencing agents can influence the flocking agents to align to θ^*, and if so, find the plan π that does so with minimum cost $c(\pi)$.

2.2.2 Placement

A k-agent placement specifies the positions that each influencing agent $\{a_0, \ldots a_{k-1}\}$ will adopt at time t_i, where time t_i is the time at which the influencing agents begin attempting to influence their neighbors. The k-agent placement is denoted by $\pi_k(t_i) = \{p_0(t_i), \ldots, p_{k-1}(t_i)\}$ where $\{p_0(t_i), \ldots, p_{k-1}(t_i)\}$ is the set of positions for influencing agents $\{a_0, \ldots a_{k-1}\}$ at time t_i.

We denote t^* as the earliest time step at which flocking agents $\{a_k, \ldots, a_{n-1}\}$ are oriented such that, for all $t \geq t^*$, $\{\theta_k(t), \ldots, \theta_{N-1}(t)\}$ are all within ϵ of θ^*. However, in some cases this cannot occur because some flocking agents may become separated from the flock — we say these agents are *lost*. An agent a_i is considered lost if two criteria hold. First, there must exist a subset of flocking agents with cardinality $0 < m' < m$ and orientations within ϵ of θ^* for more than T time steps — this means that a subset of the flock has converged to θ^* for more than T time steps. Second, $|\theta_i(t^*) - \theta^*| > \epsilon$, where t^* is the time step at which the subset converged to θ^* — this means that agent a_i did not converge to θ^* when the subset converged. In our experiments we set $T = 200$ because we experimentally found that if a set of agents remained converged to θ^* for 200 time steps, they were very likely to remain converged.

The entire flock is considered lost and the trial ends if after T_{flock} time steps no flocking agents have orientations within ϵ of θ^*. In our experiments we set $T_{\text{flock}} = 2,800$ because we have experimentally observed that influence always occurs within 2,800 time steps in our setting.

The *cost* $c(\pi(t_i))$ of a k-agent placement $\pi_k(t_i)$ is a weighted function of two terms:

- w_1 is a weight that emphasizes the importance of minimizing the number of lost agents
- w_2 is a weight that emphasizes the importance of minimizing the chances of losing any agents (as indicated by the number of simulation experiments in which any agent is lost)

$$c(\pi(0)) = w_1\overline{m - m'} + w_2\overline{p(m - m' > 0)} \tag{2.4}$$

An optimal placement $\pi^*(t_i)$ is one with minimal cost $c(\pi^*(t_i))$.

In Chapter 5, we set $w_1 > w_2$. With these preferences for w_1 and w_2 we select influencing agent placements that generally lose the fewest agents on average but that also minimize the chances of losing any agents.

Chapter 5 is focused on determining how to place influencing agents into a flock. This problem — the *Agent Placement Problem* — is stated as follows:

> Given a target orientation θ^* and a team of n agents $\{a_0, \ldots, a_{n-1}\}$, determine the desired influencing position $\pi(t_i)$ of influencing agents $\{a_0, \ldots a_{k-1}\}$ at time t_i such that cost $c(\pi(t_i))$ is minimized.

2.2.3 Joining and Leaving

In Chapter 6 we consider what would be important for a scenario in which robot birds (acting as influencing agents) leave one or more charging stations, join a flock in motion, influence the flock to avoid an airport, leave the flock, and then return to a charging station. With this scenario in mind, we consider the following metrics calculated across all of the flocking agents T_{calc} time steps after the last influencing agent left the flock (as defined by no longer influencing any flocking agents):

1. **NumIntersect**: number that intersected the airport ahead of the flock's original orientation
2. **MissionTime**: time between the first influencing agent leaving the charging station and the last influencing agent leaving the flock
3. **NumAligned**: number oriented within 10° of θ^*
4. **AvgDiff**: average difference from θ^*

In the experimental settings considered in this dissertation, we found that $T_{\text{calc}} = 2,000$ virtually guaranteed that the flocking agents were converged to their long term behaviors when the

metrics were calculated. Likewise, in our experimental setting we found that $\epsilon = 10^\circ$ allowed for differentiation between **NumAligned** and **AvgDiff**.

We consider all four metrics, but we do not define a single cost function for our joining and leaving work because the importance of each metric varies greatly based on the scenario at hand. For example, if **NumIntersect** is all that matters, then the other three metrics have no importance. However, the other metrics will usually have some importance and in some situations may be considered more important than **NumIntersect**.

2.3 Simulation Environment

This dissertation utilizes two different simulation environments: a homegrown FlockSim simulator as well as the MASON Flockers simulator. We utilized the homegrown FlockSim simulator for our early theoretical work described in Chapter 3. However, as we advanced to the more empirical and larger scale work discussed in subsequent chapters, we found that the FlockSim simulator would require significant improvements. After researching options for pre-existing flocking simulators, we opted to move forward with the MASON Flockers simulator. In this section, we describe both our FlockSim simulator as well as the MASON Flockers simulator.

2.3.1 FlockSim

Chapter 3 utilizes a homegrown custom-designed simulator FlockSim. For FlockSim experiments, the flocking agents are able to change their orientation but not their position. The influencing agents are able to adopt any orientation, but are only allowed to update their positions in some experiments. FlockSim's interface is shown in Figure 2.3.

The user first fills in the parameters along the bottom of the GUI. "Initial Flocking Angle" defines the initial orientation of any flocking agents, while "Target Angle" defines the target angle for the flocking agents at the end of the experiment. This target angle is displayed on the right in black. Both the "Initial Flocking Angle" and "Target Angle" are oriented such that 0° is directly to the east, 90° is directly to the north, and so on. "Flocking View Sector" defines the size in degrees of the flocking agent's *visibility sector*. "Number of Steps," "Velocity," and "Out

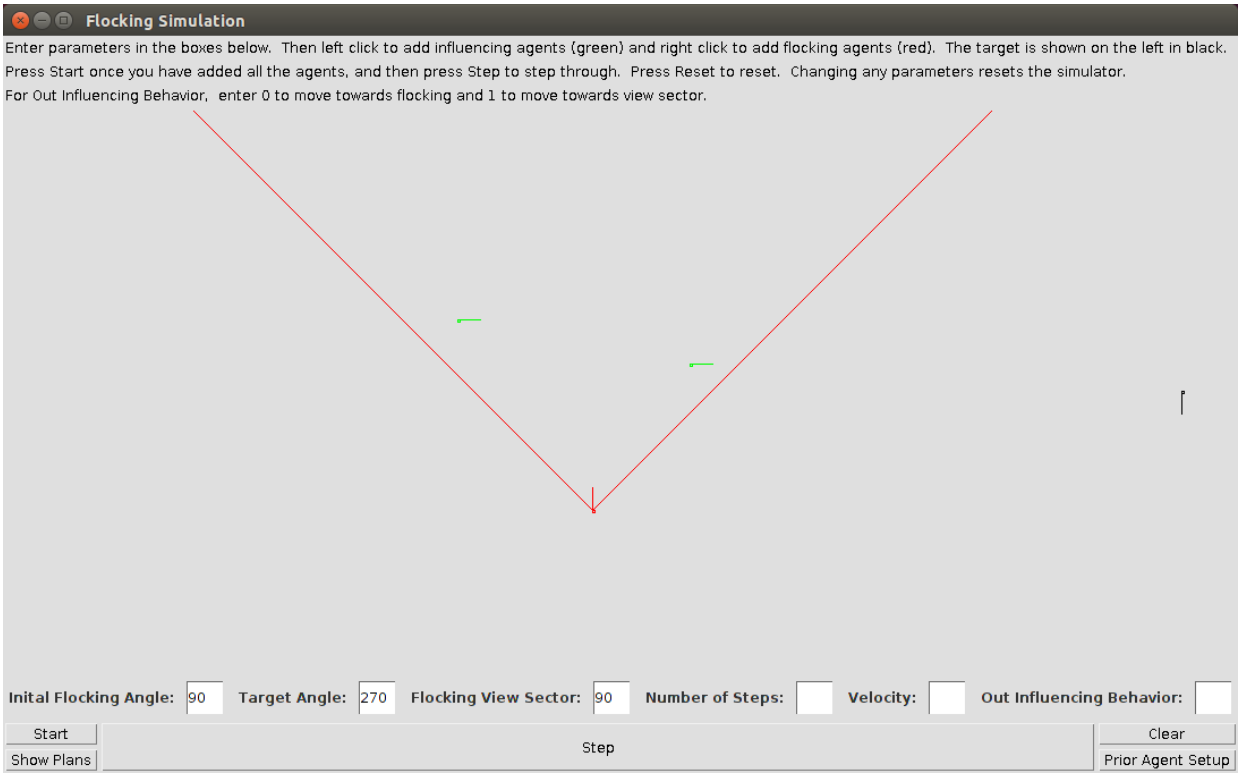


Figure 2.3: The FlockSim GUI.

Influencing Behavior" are only utilized for some experiments in which the influencing agents are able to move at each time step. In these experiments, "Number of Steps" specifies the maximum number of time steps the experiment may run, "Velocity" defines how far the influencing agent can move at each time step, and "Out Influencing Behavior" defines the influencing agent's behavior when it is able to change positions but is not currently within any flocking agents' neighborhoods.

After filling in all of the required parameters, the user should place one or more flocking agents and one or more influencing agents. Once the agents are placed, the user should start the experiment by clicking the "Start" button. Then the "Step" button should be pressed to increment though the experiment one time step at a time. During the experiment, the influencing agents will orient (and in some cases move) and the flocking agent(s) will update their orientation based on the agents currently within their neighborhood.

2.3.2 MASON Flockers

Due to our desire to scale to larger flocks, we decided to switch simulators after completing the work discussed in Chapter 3. Specifically, we situate the research in Chapters 4 – 7 of this dissertation within the Flockers domain of the MASON simulator [60]. The MASON simulator was developed at George Mason University. The creators of MASON describe it as a "fast discrete-event multiagent simulation library core in Java, designed to be the foundation for large custom-purpose Java simulations, and also to provide more than enough functionality for many lightweight simulation needs." MASON contains over 30 different domains, has high-quality online manuals, and can be downloaded for free as a 18.6KB tar.gz.[1]

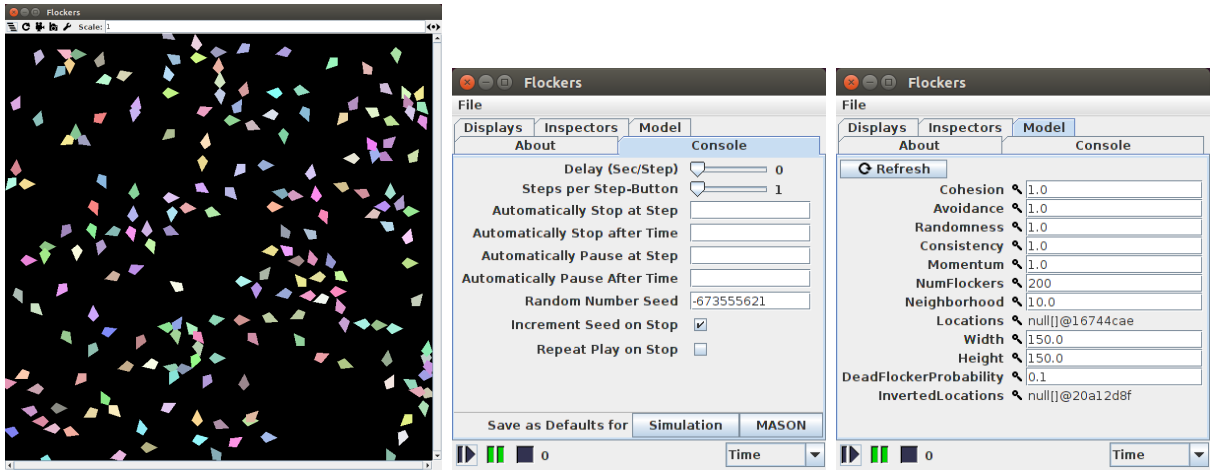
One benefit of MASON is that the models are independent from visualization, which allows for experiments to be easily and quickly executed without visualization. A second benefit we found was that initial agent positions and orientations are randomly set based upon a seed — this allows experiments to be completely reproducible as well as comparable. The MASON Flockers domain shows no significant slow down when running flocks containing hundreds of agents.

When running the visualization, the time steps often execute too quickly for individual agent behavior to be studied. However, the visualization GUI has a slide bar that can be adjusted to add delay after each time step (see Figure 2.4(b)). Additionally, the user has the ability to step through the visualization step-by-step.

In the MASON Flockers domain, each agent points and moves in the direction of its current velocity vector at each time step. As released, the MASON Flockers domain shows a chaotic swarm of agents flocking in a toroidal environment. The agents flock using a vector sum of five vectors: avoidance, cohesion, momentum, coherence, and a random vector. Together, this vector sum creates a behavior similar to Reynolds' Boid algorithm for flocking. Sample pictures of the original Flockers domain and simulator GUI are shown in Figure 2.4.

We altered the MASON Flockers domain in multiple ways for our experiments. We initially altered the MASON Flockers domain to contain *influencing agents* that follow different behaviors than the traditional flocking behaviors followed by the *flocking agents*. We also modified the *flocking agents* to only use Reynolds' *alignment* aspect as described in Section 2.1.2. Although the simulator

[1] `http://cs.gmu.edu/~eclab/projects/mason/`



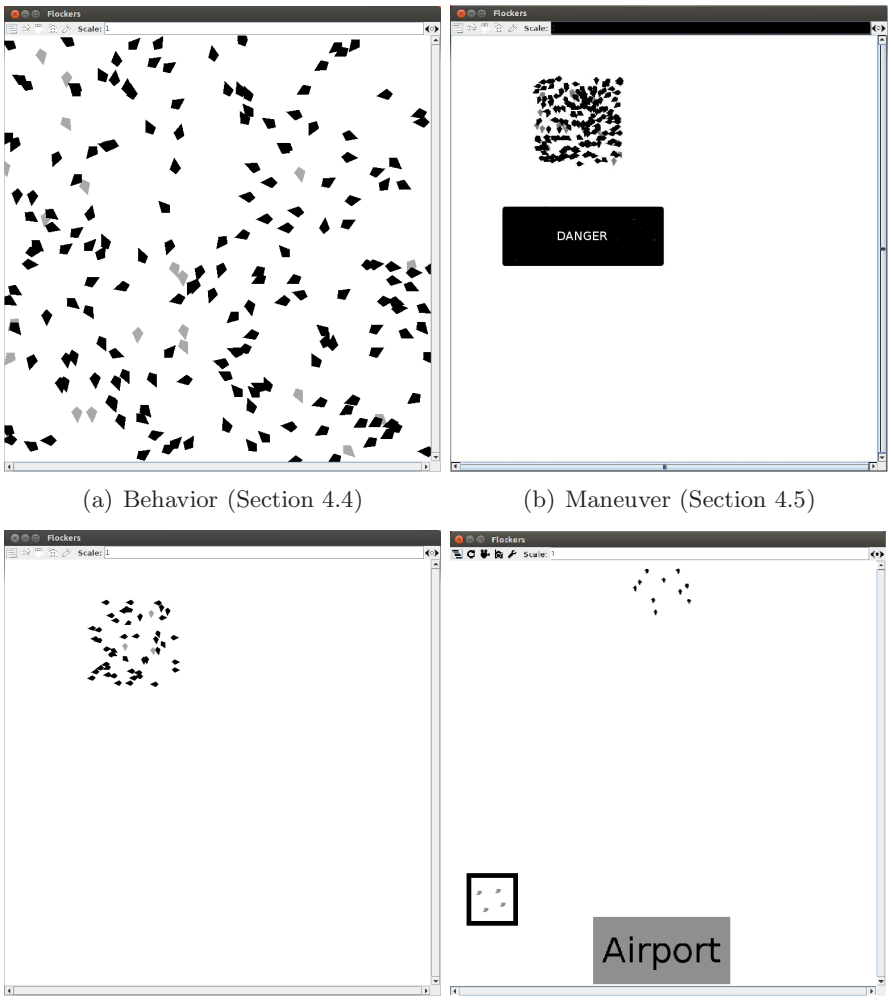
(a) Flockers visualization (b) Control Panel – Console (c) Control Panel – Model

Figure 2.4: Images of the original, unaltered MASON Flockers visualization and control panel.

was built to be toroidal — so agents that move off one edge of our domain reappear on the opposite edge moving in the same direction — we added the functionality to turn the toroidal feature of the simulator off. Once these alterations were made, we began the work described in this dissertation regarding influencing agent behavior, placement, and joining and leaving the flock.

Sample images of the MASON Flockers domain after we altered it for our experiments can be seen in Figure 2.5. Figure 2.5(a) shows gray influencing agents and black flocking agents in a toroidal domain. In this case, the influencing agents influence the flocking agents to orient towards θ^*. The environment in Figures 2.5(b–d) is non-toroidal — if an agent leaves the edge of the environment, it will not reappear on another edge. Figure 2.5(b) shows influencing agents before they influence a dense flock of flocking agents to maneuver around a dangerous area. Figure 2.5(c) shows the beginning of an experiment in which the influencing agents were placed into the flock and they must influence the flock to stay together and travel south. Finally, Figure 2.5(d) shows four influencing agents in the lower left corner and a flock of 10 flocking agent in the top center. The influencing agents must intercept the flock and influence the flocking agents to travel east instead of intersecting the airport.

Figure 2.5 provides a sneak peak of what types of experiments are covered in this dissertation. Throughout the remainder of this dissertation, we describe our specific experimental setup in more detail in each chapter before presenting that chapter's experimental results.



(a) Behavior (Section 4.4)

(b) Maneuver (Section 4.5)

(c) Placement (Chapters 5 & 7)

(d) Join (Chapter 6)

Figure 2.5: Images of the beginning of different types of experiments. The gray agents are influencing agents while the black agents are flocking agents.

3. Leading a Stationary Flock to a Desired Orientation

In this chapter, we introduce our research on leading a stationary flock of agents to a desired orientation using a subset of influencing agents.[1] We consider a stationary flock in this chapter because it allows us to introduce theoretical analysis that would be significantly more difficult — and in some cases impossible — in the more general case.

Some foundational theoretical results that apply to all flocking scenarios in this chapter are presented in Section 3.1. In Section 3.2 we set bounds on the extent of influence the influencing agents can have on the flock when both the flocking agents and the influencing agents are stationary. Section 3.3 considers the more complicated problem of orienting a stationary flock to a desired orientation using a set of non-stationary influencing agents. In Section 3.3 we provide empirical evaluations using our custom-designed simulator FlockSim that was described in Section 2.3.1.[2]

The work in this chapter assumes that each agent utilizes the *visibility sector* neighborhood model described in Section 2.1.1 and the *Flock Manipulation* performance metric described in Section 2.2.1. To simplify our theoretical analysis, in this chapter we assume that all flocking agents are located at a single identical position p_i and orientation $\theta_i(t)$ in the environment. Hence, although the flocking agents' orientation is dependant on t, their position is not.

3.1 General Flocking Theorems

In this section, we present lemmas that are general in nature and will apply to both the stationary and non-stationary influencing agent cases examined in the later sections of this chapter. In particular, following the notation introduced in Section 2.1.1, we consider the case in which there are $k_i(t)$ influencing agents within the neighborhood of $m_i(t)$ flocking agents. In this case, all $m_i(t)$

[1] This chapter is based on a conference paper [31] that I wrote with Noa Agmon and Peter Stone. Author contributions were as follows: I was a Ph.D. student and did the complete implementation and writing. Peter was my advisor and Noa was a post-doctorate fellow in Peter's group. Peter and Noa both collaborated with me on deciding research directions and interpreting results. Noa also collaborated with me on completing proofs.

[2] Videos of our FlockSim simulator are available at `http://www.cs.utexas.edu/~katie/videos/`

flocking agents are located at the same position p_i with identical orientations $\theta_i(t)$ (see Figure 3.1 for an example). Recall from Section 2.1.1 that under the *visibility sector* neighborhood model, the $m_i(t)$ flocking agents are included in their own neighborhoods.

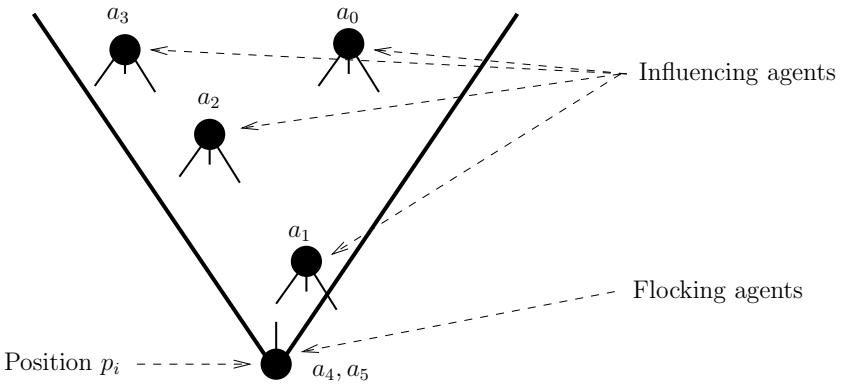


Figure 3.1: An example with two flocking agents located at the same position with identical initial orientations and four influencing agents located at different locations within the visibility sector of the flocking agents.

The first lemma we present in this section relates to the maximal amount the $k_i(t)$ influencing agents can influence the $m_i(t)$ flocking agents in a single time step.

Lemma 1 *The $k_i(t)$ influencing agents can influence the $m_i(t)$ flocking agents to turn in a particular direction by any amount less than or equal to $\frac{k_i(t)\pi}{m_i(t)+k_i(t)}$ radians in one time step.*

Proof When the difference between $\theta_j(t)$ and $\theta_i(t)$ is less than π (or greater than π, in which case the difference is less than π in the opposite direction), then by Equation 2.1

$$\begin{aligned}
\theta_i(t+1) - \theta_i(t) &= \frac{1}{n_i(t)} \sum_{j \in N_i(t)} (\theta_j(t) - \theta_i(t)) \\
&\leq \frac{k_i(t)(\pi - \epsilon)}{m_i(t) + k_i(t)} \\
&\leq \frac{k_i(t)\pi}{m_i(t) + k_i(t)} - \frac{k_i(t)\epsilon}{m_i(t) + k_i(t)} \\
&< \frac{k_i(t)\pi}{m_i(t) + k_i(t)}
\end{aligned}$$

When the difference between $\theta_j(t)$ and $\theta_i(t)$ is equal to π, by Equation 2.1

$$\begin{aligned}\theta_i(t+1) - \theta_i(t) &= \frac{1}{n_i(t)} \sum_{j \in N_i(t)} (\theta_j(t) - \theta_i(t)) \\ &= \frac{k_i(t)\pi}{m_i(t) + k_i(t)}\end{aligned}$$

However, it is impossible to guarantee that the flocking agents turn in a particular direction when the difference between $\theta_j(t)$ and $\theta_i(t)$ is equal to π. Hence, in this case the influencing agents set $\theta_j(t)$ such that the difference between $\theta_j(t)$ and $\theta_i(t)$ is $\pi - \epsilon$ or $\pi + \epsilon$. When the influencing agents do this, directionality can be guaranteed and

$$\begin{aligned}\theta_i(t+1) - \theta_i(t) &= \frac{k_i(t)(\pi - \epsilon)}{m_i(t) + k_i(t)} \\ &< \frac{k_i(t)\pi}{m_i(t) + k_i(t)}\end{aligned}$$

□

The second lemma we present in this section states that all $k_i(t)$ influencing agents in a flocking agent's visibility sector can adopt the exact same orientation and still optimally influence the flock. Specifically, we show that no extra influence can be obtained by some of the influencing agents adopting different orientations than the other influencing agents.

Lemma 2 *When $k_i(t)$ influencing agents work together to influence $m_i(t)$ flocking agents to align the team to some θ, it suffices to consider only algorithms that choose at each time step just one orientation for all of the influencing agents to adopt.*

Proof Assume an algorithm makes the influencing agents adopt orientations $\theta_0(t), \ldots, \theta_{k_i(t)-1}(t)$, where some of these orientations may differ. Then, by Equation 2.1, the orientation of the flocking agents is

$$\theta_f(t+1) = \theta_f(t) + \frac{1}{n_f(t)} \sum_{j \in N_i(t)} \text{calcDiff}(\theta_j(t), \theta_f(t))$$

Now, assume the influencing agents adopt an angle σ that is the average of $\theta_0(t), \ldots, \theta_{k_i(t)-1}(t)$.

Then by Equation 2.1, the new orientation is

$$\theta_f(t+1) = \theta_f(t) + \frac{k_i(t)}{n_f(t)}(\sigma)$$

Since σ is the average of $\theta_0(t), \ldots, \theta_{k_i(t)-1}(t)$,

$$\frac{1}{n_f(t)} \sum_{j \in N_i(t)} \text{calcDiff}(\theta_j(t), \theta_f(t)) = \frac{k_t(i)}{n_t(f)}(\sigma)$$

Therefore, for every algorithm assigning different orientations, there is some algorithm assigning the same orientation, which concludes the proof. □

3.2 Stationary Agents

In this section we consider the case in which there are $m_i(t)$ flocking agents located at a single position p_i with identical initial orientations, $k_i(t)$ influencing agents located at arbitrary locations within the flocking agents' neighborhood at time t, and $k - k_i(t)$ influencing agents located at arbitrary locations outside of the flocking agent's neighborhood at time t. Each agent a_i has velocity $v_i = 0$. This means that although an agent's orientation may change, its position will remain constant. An example is provided in Figure 3.2.

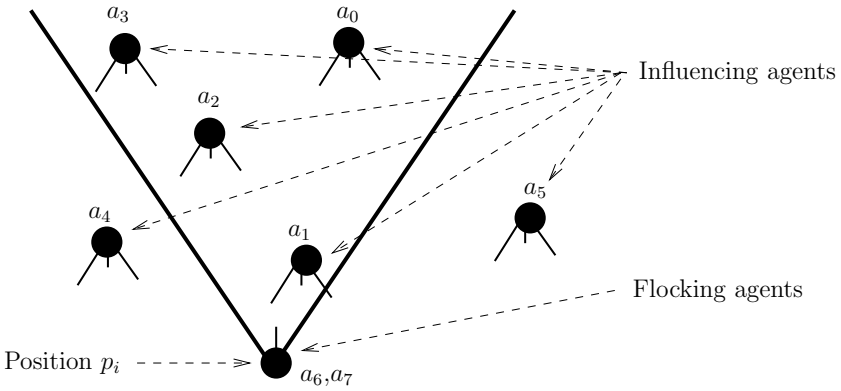


Figure 3.2: An example with two flocking agents (a_6 and a_7) located at the same position with identical initial orientations, four influencing agents (a_0, a_1, a_2, and a_3) located at different locations within the visibility sector of the flocking agents, and two influencing agents (a_4 and a_5) located at different locations outside the current visibility sector of the flocking agents.

As the flocking agents are influenced to turn towards θ^*, different influencing agents become available to influence the flocking agents. This is because some influencing agents may no longer be within the flocking agents' visibility sector, while other influencing agents may enter the visibility sector. Hence, at each time step the influencing agents must consider the trade-off between moving the flocking agents maximally towards θ^* and keeping influencing agents within the flocking agents' visibility sector for future time steps.

In this section, we introduce some new terminology: *border agent*, *border influence orientation*, and $\beta_j(i)$.

A *border agent* is an influencing agent that is located within the visibility sector of the flocking agents, on the edge of the visibility sector that is farther away from the target. A *border influence orientation* is a flocking agent orientation at which an influencing agent is a border agent. Clearly for each influencing agent, there are exactly two possible border influence orientations — one in which the border agent is located on the left hand side of the flocking agents' visibility sector and one in which the border agent is located on the right hand side of the visibility sector. See Figure 3.3 for an example with a border agent.

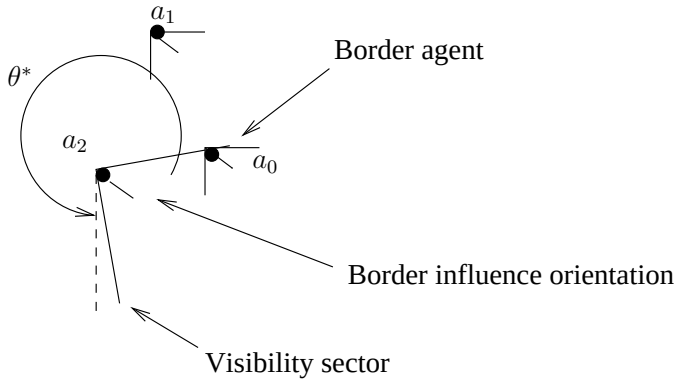


Figure 3.3: An example of a border agent (a_0) and the resulting border influence orientation of the flocking agent (a_2).

Recall that α denotes the angle of the flocking agent a_i's visibility sector. Agent a_j's position $p_j(t) = (x_j(t), y_j(t))$ in the environment at time t is located at angle $\beta_j(i)$ with respect to agent a_i's position $p_i(t) = (x_i(t), y_i(t))$ and orientation $\theta_i(t)$. Agent a_j is in a_i's neighborhood at time t if angle $\beta_j(i)$ is less than or equal to $\frac{\alpha}{2}$. Figure 3.4 demonstrates this concept.

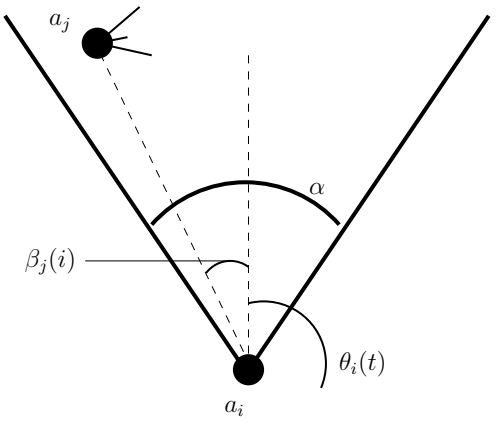


Figure 3.4: Angle α defines the visibility sector for agent a_i. Agent a_j is in a_i's neighborhood since angle $\beta_j(i) \leq \frac{\alpha}{2}$.

The first lemma in this section puts a bound on the maximal amount the flocking agents can be influenced to turn and still have the same set of influencing agents and flocking agents within the flocking agents' visibility sector.

Lemma 3 *$k_i(t)$ influencing agents within the neighborhood of $m_i(t)$ flocking agents can influence the $m_i(t)$ flocking agents to turn $min(\beta_j(i)+\frac{\alpha}{2}, \frac{k_i(t)\pi}{m_i(t)+k_i(t)}-\epsilon)$ radians in one time step and still have the same $m_i(t)$ flocking agents and $k_i(t)$ influencing agents within the flocking agents' neighborhood.*

Proof In order for all the $k_i(t)$ agents to remain in the neighborhood of all $m_i(t)$ agents at time $t+1$, it is necessary for the amount the $m_i(t)$ flocking agents turn by $(\min(\beta_j(i)+\frac{\alpha}{2}, \frac{k_i(t)\pi}{m_i(t)+k_i(t)}-\epsilon))$ plus the location of the current edge of the flocking agents' visibility sector $(\theta_i(t) - \frac{\alpha}{2})$ to be less than or equal to the orientation of the position of the influencing agents with respect to the position of the flocking agents $(\beta_j(i) + \theta_i(t))$. Hence,

$$\min(\beta_j(i) + \frac{\alpha}{2}, \frac{k_i(t)\pi}{m_i(t) + k_i(t)} - \epsilon) + \theta_i(t) - \frac{\alpha}{2} \leq \beta_j(i) + \theta_i(t) \tag{3.1}$$

when $m_i(t)$ flocking agents and $k_i(t)$ influencing agents are in each flocking agents' neighborhood at time $t + 1$.

If $\beta_j(i) + \frac{\alpha}{2} < \frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$, then $\beta_j(i) + \frac{\alpha}{2} + \theta_i(t) - \frac{\alpha}{2} \leq \beta_j(i) + \theta_i(t)$. The left side of Equation 3.1 clearly equals the right side in this case.

Otherwise, if $\beta_j(i) + \frac{\alpha}{2} \geq \frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$, then $\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon + \theta_i(t) - \frac{\alpha}{2} \leq \beta_j(i) + \theta_i(t)$. Since, $\beta_j(i) + \frac{\alpha}{2} \geq \frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ in this case, the left side of Equation 3.1 is less than or equal to the right side. □

The second lemma in this section sets a bound on the maximum number of time steps needed for the influencing agents to influence the flocking agents to reach θ^* when θ^* is reachable.

Lemma 4 *The $k_i(t)$ influencing agents can influence the $m_i(t)$ flocking agents to align the team to θ^* within*

$$Z = 1 + \lceil \frac{min(\frac{\pi}{2}, \alpha)}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon} \rceil$$

time steps when θ^* is reachable (i.e. the difference between $\theta_i(t)$ and θ^* is less than or equal to $(Z-1)\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon + \beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2}$ and $\alpha > (Z-2)\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$).

Proof By Lemma 1, $k_i(t)$ influencing agents can influence $m_i(t)$ flocking agents to turn by $\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ on each of the first $Z-2$ time steps. Additionally, by Lemma 3, $k_i(t)$ influencing agents can influence $m_i(t)$ flocking agents to turn by $\beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2}$ on the $Z-1$ time step and still have have $m_i(t)$ flocking agents and $k_i(t)$ influencing agents in each flocking agents' neighborhood. Finally, by Lemma 1 $k_i(t)$ influencing agents can influence $m_i(t)$ flocking agents to turn by any amount less than or equal to $\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ on the last time step.

Influencing as described above must force the $m_i(t)$ flocking agents to align to θ^*; in other words, we must show that

$$(Z-2)\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon + \beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2} + \frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon \geq \pi \qquad (3.2)$$

By definition we know that $\alpha > (Z-2)\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ and $\alpha \leq 2\pi$, so the left side of Equation 3.2 simplifies to $3\pi + \beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{2\pi}{Z-2}$ such that the left side of Equation 3.2 is greater than or equal to the right side. □

The following theorem states that it is impossible to influence the flocking agents to orient themselves to θ^* (assuming it is reachable) in fewer than Z time steps. Remember that Lemma 4

showed that $k_i(t)$ influencing agents can influence $m_i(t)$ flocking agents to align the team to θ^* in Z time steps when θ^* is reachable.

Theorem 1 *If alignment is possible,*

$$Z = 1 + \lceil \frac{min(\frac{\pi}{2}, \alpha)}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon} \rceil$$

time steps are needed for the $k_i(t)$ influencing agents to influence the $m_i(t)$ flocking agents to align the team to θ^*.

Proof Assume, towards contradiction, that there exists an algorithm in which $k_i(t)$ influencing agents influence $m_i(t)$ flocking agents to align the team to θ^* (when alignment is possible) in $Z' < Z$ time steps.

By Lemmas 1, 3, and 4, $k_i(t)$ influencing agents can influence $m_i(t)$ flocking agents to turn $\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ on each of the first $Z' - 2$ time steps, by $\beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2}$ on the $Z' - 1$ time step, and by at most $\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ on time step Z'. Hence,

$$(Z' - 1)\frac{k_i(t)\pi}{m_i(t) + k_i(t)} - \epsilon + \beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2} \geq \pi \tag{3.3}$$

when alignment of the team to θ^* can be achieved in Z' time steps. By Lemmas 1 and 3, $\beta_j(i) + \theta_i(t) - \theta_i(t+1) + \frac{\alpha}{2} \leq \frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon$ so the left side of Equation 3.3 becomes $Z'(\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon)$.

If $\frac{\pi}{2} > \alpha$, then

$$Z' \leq \frac{\alpha}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon}$$

In this case, the left of Equation 3.3 becomes

$$(\frac{\alpha}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon})(\frac{k_i(t)\pi}{m_i(t) + k_i(t)} - \epsilon) = \alpha = \frac{\pi}{2}$$

Otherwise, if $\frac{\pi}{2} \leq \alpha$, then

$$Z' \leq \frac{\frac{\pi}{2}}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon}$$

In this case, the left of Equation 3.3 becomes

$$(\frac{\frac{\pi}{2}}{\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon})(\frac{k_i(t)\pi}{m_i(t)+k_i(t)} - \epsilon) = \frac{\pi}{2} < \pi$$

leading to a contradiction. □

When determining how the influencing agents should orient themselves to optimally influence the flocking agents, we use a *forward search* approach (see Figure 3.5). Specifically, beginning at the initial flocking orientation, we consider each possible border influence orientation. If the border influence orientation is reachable from the initial flocking orientation, then we consider each possible border influence orientation from this point. If the border influence orientation is not reachable from the initial flocking orientation, then we turn to the farthest reachable point and then determine if the border influence orientation is now reachable (and repeat this process until the border influence orientation is reachable). We repeat this process until the target is within reach, and we select the plan that reaches the target in the fewest number of steps.

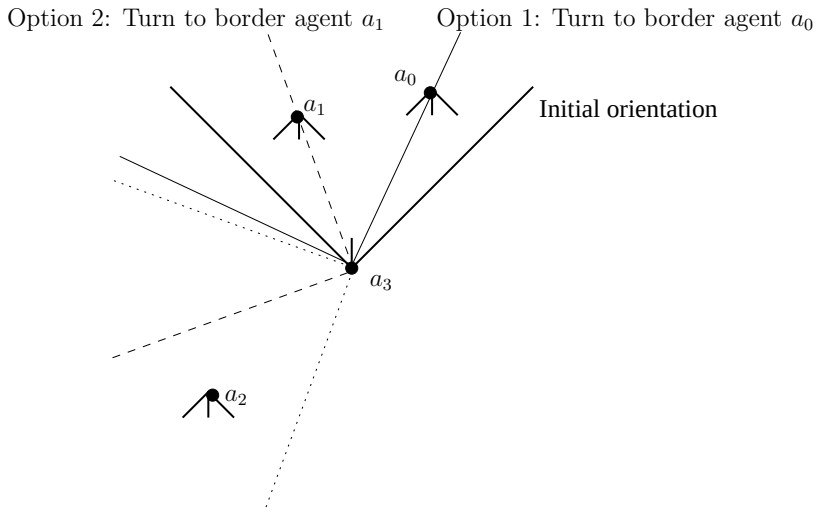


Figure 3.5: An example of the possible subsequent flocking agent orientations for a given initial orientation. The influencing agents are labelled with a_0, a_1, and a_2, while the flocking agent is labelled with a_3. Note that turning to border agent a_2 is not possible in the first time step.

A forward search such as this requires checking 2^k possible combinations of the number of influencing agents influencing the flocking agents at each time step. Consider the case where

Variable	Definition
bestFSeq	the flocking sequence of orientations that uses the least number of time steps to reach θ^*
bestIAPlan	the influencing agent plan that uses the least number of time steps to reach θ^*
borderTarget	the border influence orientation needed to be a border agent
ccw	whether the flocking agents are rotating counter-clockwise
current	the orientation the flocking agents are currently oriented towards
currentIAPlan	the plan containing the orientations for each influencing agent at each time step so far
currentFSeq	the sequence of orientations for the flocking agents at each time step so far
inflOrient	the orientation the influencing agents must adopt at this time step in order for the flocking agents to reach *target* from *current*
initFOrient	the initial orientation of the flocking agents
maxSteps	the maximum number of steps a plan can be
numF	the number of flocking agents
numIA	the number of influencing agents within the flocking agents' visibility sector
target	the orientation the flocking agents should be oriented towards on the next time step
targetReachable	whether *target* is reachable from *current*

Table 3.1: Variables used in Algorithm 2.

there are three influencing agents. The following eight combinations of targets covers all possible combinations: $[a_0, a_1, a_2], [a_0, a_1], [a_0, a_2], [a_0], [a_1, a_2], [a_1], [a_2], []$. By convention, agent a_0 will be oriented farther from the target than agent a_1, which will be oriented farther from the target than agent a_2, and so on. See Figure 3.6 for an example with two influencing agents and one flocking agent.

Algorithm 2 uses such a forward search approach to calculate and return (1) the number of steps needed to reach θ^* and (2) the necessary orientations for each of the influencing agents for each of these steps. Throughout the algorithm, element.get(x) returns the 0-indexed x item in element (where element is a list object), element.add(y) adds item y to the end of element, and element.size() returns the number of items contained in element. The variables used throughout Algorithm 2 are defined in Table 3.1. Remember that although the influencing agents are located at many arbitrary locations, the $m_i(t)$ flocking agents are located at a single position p_i and begin with identical orientations.

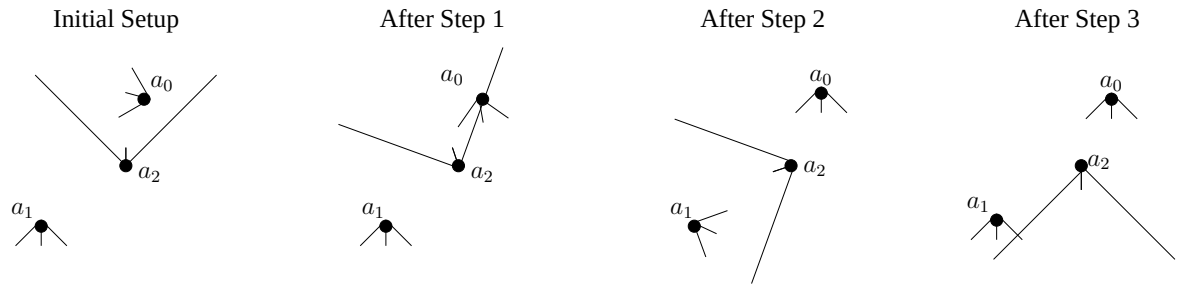


Figure 3.6: Example with two influencing agents (a_0 and a_1) and one flocking agent (a_2). On step 1, the flocking agent turns to have a_0 as a border agent. Then on step 2, the flocking agent turns as much as possible towards θ^*. Finally, on step 3 the flocking agent turns to θ^*.

Algorithm 2 plan, steps = calcPlan()

```
1:  for each possible influencing agent combination do
2:    currentFSeq, currentIAPlan ← ()
3:    current ← initFOrient
4:    for each borderTarget in this influencing agent combination do
5:      targetReachable ← false
6:      while targetReachable == false and currentFSeq.size() < maxSteps and bestFSeq.size() > currentFSeq.size() do
7:        target ← borderTarget
8:        if ccw then
9:          target ← target + α/2
10:       else
11:         target ← target − α/2
12:       if |calcDiff(current, target)| > numIAπ/(numIA+numF) then
13:         targetReachable ← false
14:         if ccw then
15:           target ← current + numIAπ/(numIA+numF) − ϵ
16:         else
17:           target ← current − numIAπ/(numIA+numF) + ϵ
18:       else
19:         targetReachable ← true
20:       inflOrient ← |calcDiff(target, current)|∗(numF+numIA)/numIA + target
21:       currentFSeq.add(target)
22:       for each influencing agent x in the flocking agents' visibility sector when facing current do
23:         currentIAPlan.get(x).add(inflOrient)
24:       current ← target
25:     end while
26:   if currentFSeq is smaller than bestFSeq then
27:     bestFSeq ← currentFSeq
28:     bestIAPlan ← currentIAPlan
29: return bestIAPlan, bestFSeq.size()
```

Now we consider Algorithm 2's forward search approach in detail. Lines 1–28 consider each possible combination of influencing agents that could influencing the flocking agents. For each combination, lines 4–25 consider each *borderTarget* and build a plan of influencing agent orientations *currentIAPlan*. Specifically, lines 6–25 build *currentIAPlan* while (1) *target* is not reachable, (2) the current sequence of orientations for the flocking agents *currentFSeq* is not too long, or (3) *bestFSeq* is longer than *currentFSeq*. The current plan is built by updating *target* (lines 7–11), determining if *target* is reachable (line 12), and then updating *target* to be reachable if it was unreachable (lines 13–17). Now that *target* is reachable, line 20 calculates the required influencing agent orientation *inflOrient* for *target* to be reached by the flocking agents. At this point, line 21 adds *target* to *currentFSeq* and lines 22 and 23 add *inflOrient* to *currentIAPlan* for each influencing agent in the flocking agents' neighborhood. Line 24 updates the current flocking agent orientation *current* to *target*. Lines 26–28 update *bestFSeq* and *bestIAPlan* when *currentFSeq* is shorter than *bestFSeq*. Finally, line 29 concludes the algorithm by returning *bestIAPlan* and the size of number of time steps required to orient the flocking agents towards θ^*.

In the worst case, line 1 will be executed 2^{numIA} times, line 4 will execute $\text{numIA} + 1$ times, and line 6 will execute maxSteps times. Hence, lines 7–24 are executed at most $(2^{\text{numIA}})(\text{numIA} + 1)(\text{maxSteps})$ times.

Theorem 2 *Given θ^* and assuming the $m_i(t)$ flocking agents are influenced only by the $k_i(t)$ influencing agents and the $m_i(t)$ flocking agents at time t, then if θ^* is reachable, the influencing agents are guaranteed to lead the flocking agents to θ^* in the least number of time steps possible when the influencing agents determine their plan based on Algorithm 2 and the number of steps required is not larger than maxSteps.*

Proof There are exactly 2^{numIA} possible influencing agent combinations. Hence, by line 1, Algorithm 2 is guaranteed to consider each possible influencing agent combination.

Each border influence orientation target is considered until it is reachable or the plan size becomes larger than maxSteps (line 6), and necessary influencing agent orientations are added to the currentIAPlan and targets are added to the currentFSeq until it is reachable or the plan size becomes larger than maxSteps. The currentFSeq and currentIAPlan become the bestFSeq and

bestIAPlan (lines 25–27) only if they use less steps to reach θ^* than the current bestFSeq and bestIAPlan. Line 25 will not be reached until all border orientation targets for a particular set of influencing agent combinations have been considered and θ^* has been reached or the plan size becomes larger than maxSteps. Hence, since the best possible influencing agent combinations are guaranteed to be considered and the number of steps required will not be larger than maxSteps, we are guaranteed that the bestIAPlan that is returned by Algorithm 2 is the least-step plan possible. □

Algorithm 2 has been implemented and tested in our custom-designed simulator FlockSim. Results from experiments using FlockSim are given in Section 3.3.

3.3 Non-stationary Influencing Agents

In this section we consider the case in which there are $m_i(t)$ flocking agents that are all located at position p_i with identical initial orientations. These flocking agents remain stationary at position p_i, but may change orientation if influenced by at least one influencing agent. k influencing agents that travel with a constant velocity are initially located at arbitrary locations throughout the environment. As introduced in Section 2.1.2, each influencing agent's position $p_i(t) = (x_i(t), y_i(t))$ is updated during each time step after its orientation is updated. Hence, $x_i(t) = x_i(t-1) + v_i(t)\cos(\theta_i(t))$ and $y_i(t) = y_i(t-1) - v_i(t)\sin(\theta_i(t))$. The flocking agents, on the other hand, have velocity $v_i(t) = 0$ in this section because they are stationary.

In Section 3.2, the main decision for each influencing agent was whether to influence the flocking agents to turn maximally towards θ^* or to influence the flocking agents to turn such that one of the influencing agents becomes a border agent. However, determining exactly how non-stationary influencing agents should behave is a more complicated problem. Hence, in this section we consider some heuristic approaches for how non-stationary influencing agents should behave when influencing stationary flocking agents.

In the stationary influencing agents case, it did not matter how the influencing agents that were not within any flocking agent visibility sectors behaved because they had no influence over any flocking agents. However, non-stationary influencing agents travel in the direction they are

facing, so it does matter what orientation they face even when they are not within the visibility sector of any flocking agents. Hence, in this work we present two heuristic behaviors for influencing agents that are *not* within the visibility sector of any flocking agents: *Towards Visibility Sector* and *Towards Flocking Agent*.

We provide algorithms for the *Towards Visibility Sector* and *Towards Flocking Agent* in this section. Each influencing agent uses these algorithms to determine its behavior when it is not within the neighborhood of any flocking agents. We specify the inputs to the algorithms here for clarity. *flockingLoc* refers to the location in the environment of the flocking agents, while *influencingLoc* refers to the location of the influencing agent. *flockingOrient* refers to the current orientation of the flocking agents, while *flockingTarget* refers to the expected orientation of the flocking agents at the next time step. Finally, *influencingVelocity* is the velocity of the influencing agent.

Towards Visibility Sector orients each influencing agent towards the closest point on the flocking agents' visibility sector based on each influencing agent's current position. Algorithm 3 describes how this orientation towards the visibility sector is calculated. Line 3 calculates the orientation that points directly towards the flocking agent — this orientation is returned when line 6 calculates that the influencing agent is either very near to the flocking agent or its' visibility sector. Otherwise, line 8 determines on which side of the flocking agents' visibility sector the influencing agent is positioned. Lines 9 and 11 return orientations on each side of the visibility sector that allow the influencing agents to orient towards the closest point on the visibility sector. Note that lines 6, 8, and 9 utilize the calcDiff algorithm detailed in Algorithm 1.

Algorithm 3 angleToAdopt = towardsVisibilitySector(flockingLoc, influencingLoc, flockingOrient, flockingTarget)

1: x1 $\leftarrow$ flockingLoc.x-influencingLoc.x
2: y1 $\leftarrow$ influencingLoc.y-flockingLoc.y
3: relativeAngle $\leftarrow$ arctan (y1,x1)
4: x2 $\leftarrow$ influencingLoc.x-flockingLoc.x
5: y2 $\leftarrow$ flockingLoc.y-influencingLoc.y
6: **if** $\alpha < \pi$**and** calcDiff(arctan (y2,x2), flockingOrient) $> \frac{\pi}{2} + \frac{\alpha}{2}$ **then**
7: **return** relativeAngle
8: **else if** calcDiff(relativeAngle, flockingTarget) > 0 **then**
9: **return** calcDiff(flockingTarget, $\frac{\alpha}{2}$) + $\frac{\pi}{2}$
10: **else**
11: **return** flockingTarget + $\frac{\alpha}{2} - \frac{\pi}{2}$

Towards Flocking Agent orients the influencing agent towards the flocking agent's position. Algorithm 4 describes how the orientation towards the flocking agent is calculated. In this algorithm, line 3 calculates the orientation that would point directly towards the flocking agent. This orientation will usually be returned on line 13. However, line 6 calculates the distance between the influencing agent's location and the closest intersection point with the flocking agents' current visibility sector. Line 8 checks whether the distance calculated on line 6 can be covered in one time step and whether any other influencing agents are within the flocking agents' current neighborhood. If the distance can be covered in one time step and no other influencing agents are influencing the flocking agents, then line 9 returns an orientation towards the flocking agent's current visibility sector. Otherwise, line 7 calculates the distance between the influencing agent's location and the closest intersection point with the flocking agents' expected visibility sector on the next time step. Line 10 checks whether the distance calculated on line 7 can be covered in one time step and whether any other influencing agents are within the flocking agents' current neighborhood. If the distance can be covered in one time step and other influencing agents are influencing the flocking agents during the current time step, then line 11 returns an orientation towards the flocking agent's expected visibility sector at the next time step.

Algorithm 4 angleToAdopt = towardsFlockingAgent(flockingLoc, influencingLoc, flockingOrient, flockingTarget, influencingVelocity)

1: x $\leftarrow$ flockingLoc.x-influencingLoc.x
2: y $\leftarrow$ influencingLoc.x-flockingLoc.x
3: angleToAdopt $\leftarrow$ arctan (y, x)
4: x2 $\leftarrow$ influencingLoc.x-flockingLoc.x
5: y2 $\leftarrow$ flockingLoc.y-influencingLoc.y
6: distToViewsector $\leftarrow \sin(\text{flockingOrient} - \frac{\alpha}{2} - \arctan(\text{y2,x2})) * \sqrt{x2^2 + y2^2}$
7: distToNewViewsector $\leftarrow \sin(\text{flockingTarget} - \frac{\alpha}{2} - \arctan(\text{y2,x2})) * \sqrt{x2^2 + y2^2}$
8: **if** distToViewsector $<$ influencingVelocity **and** numAdHoc $== 0$ **then**
9: **return** towardsVisibilitySector(flockingLoc, influencingLoc, flockingOrient, flockingOrient)
10: **else if** distToNewViewsector $<$ influencingVelocity **and** numAdHoc > 0 **then**
11: **return** towardsVisibilitySector(flockingLoc, influencingLoc, flockingOrient, flockingTarget)
12: **else**
13: **return** angleToAdopt

The performance of each of these behaviors is studied empirically later in this section and reported on in Figure 3.7. Analysis of these behaviors is empirical instead of theoretical because

it becomes very difficult to prove optimality once the influencing agents have non-zero velocity. Throughout the remainder of this chapter, remember that these behaviors are not provably optimal, but instead heuristics.

Although one of these behaviors must currently be chosen by the user for each trial, the optimal behavior likely consists of some combination of these behaviors and perhaps other behaviors. The exact situations in which each behavior should be utilized have not been determined, but there are some situations where each behavior may be best. Moving towards the visibility sector may be ideal when no influencing agents are currently in the visibility sector to influence the flocking agent, as the flocking agent will not be able to be influenced until at least one influencing agent moves within its' visibility sector. On the other hand, moving towards the flocking agent may be ideal when there are influencing agents currently within the flocking agents' visibility sector, as moving closer to the flocking agent now will decrease the number of time steps required for the influencing agent to enter the flocking agents' visibility sector in future time steps.

The general behavior for non-stationary influencing agents that are inside a flocking agent's visibility sector is similar to the behavior of stationary influencing agents. Specifically, the non-stationary influencing agents will either influence the flocking agents to turn maximally or they will influence the flocking agents to turn such that an influencing agent is at the edge of the visibility sector (and hence a border agent). The main difference between the stationary influencing agents behavior and the non-stationary influencing agents behavior is that now the border agent must be on the edge of the visibility sector after updating its location. There are exactly two orientations at which a non-stationary influencing agent can orient and be a border agent. One of these orientations results in the influencing agent becoming a border agent on the left hand side of the visibility sector, while the other orientation results in the influencing agent becoming a border agent on the right hand side of the visibility sector. There must be exactly two orientations at which a non-stationary influencing agent can orient and be a border agent because any other orientations will result in either the flocking agents being influenced to turn farther and the influencing agent no longer moving enough to move into the visibility sector or in the flocking agents not being influenced to turn as much and the influencing agent being too far inside the visibility sector to be a border agent. We could find the exact orientation at which a non-stationary influencing agent can orient and be

a border agent by performing a binary search for the exact orientation. However, we instead use a simpler, more efficient heuristic approach that finds an orientation close to the exact orientation that would be found by the binary search such that the influencing agent is still within the flocking agents' visibility sector after moving.

Non-stationary influencing agents clearly have more influence than stationary influencing agents. In some situations, convergence of the flocking agents to θ^* is able to occur quicker. In other situations, non-stationary influencing agents are able to lead the flocking agents to converge to θ^* in cases where stationary influencing agents would be unable to. There are situations in which an influencing agent can travel into the flocking agent's visibility sector and influence when it would have been unable to influence if it were stationary.

Empirical Evaluation

All of the heuristic behaviors described in this section have been implemented and tested in FlockSim. Earlier in this section we presented two heuristic behaviors for influencing agents that are located outside of the flocking agents' visibility sector. Now we examine each of these behaviors in FlockSim, and study (1) is there a significant difference in the number of steps required for the flocking agents to orient to θ^* with each heuristic behavior and (2) how well do our influencing agents perform when compared with the naive method used by others (e.g. [50, 75]) in which the controllable agents orient towards θ^* such that the flock slowly converges to θ^*?

The results of our experiments are presented in Figure 3.7. For these experiment, $v = 50$ for the influencing agents, $\alpha = 90°$, $\theta^* = 270°$, the initial flocking orientation was 90°, and the influencing agents and flocking agents were placed randomly in a 950 by 500 environment. Each of the runs within the three possible team configurations used the same randomization seed. maxSteps was set to 100, such that no trials stopped due to the plan size exceeding maxSteps. When run with teams composed of one to four non-stationary influencing agents and one to four stationary flocking agents on a Dell Precision-360 desktop computer, an optimal plan is found in 0.0037 seconds on average.

As seen in Figure 3.7, **Towards Visibility Sector** performs significantly better than **Towards Flocking Agent** in all the configurations utilizing influencing agents. **Towards Visibility**

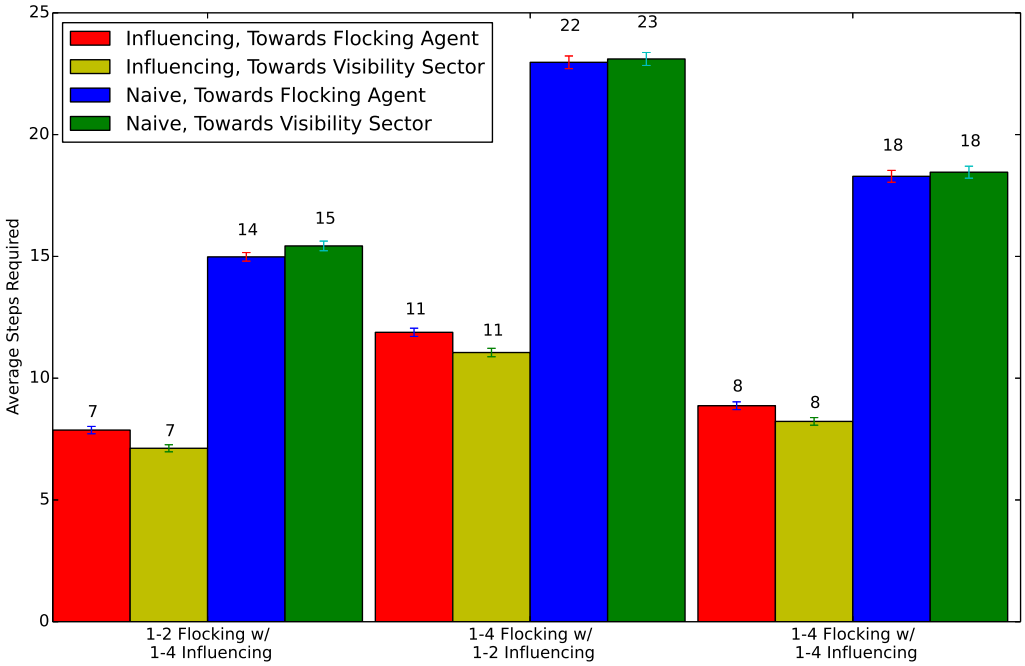


Figure 3.7: Results obtained using FlockSim on three different team configurations over 1000 trials with default settings. The error bars depict the standard error of the mean.

Sector likely performed better because getting into the visibility sector faster allows the influencing agent to influence the flocking agent sooner.

Figure 3.7 also shows that our influencing agent algorithms perform significantly better than the naive method in which the controllable agents orient towards θ^*. Our influencing agent algorithms performed better because we purposely orient the influencing agents past θ^* in order to orient the flocking agents exactly to θ^* quickly. It is important to note that in this experiment we relaxed the definition of "reaching" θ^* for the naive method. Due to the way in which the naive method slowly converges, under our strict definition of "reaching," the naive methods would very rarely converge.

As evidence that the results discussed in this section hold across different experimental settings, we considered a different value of α as well as a different velocity v. Specifically, Figure 3.8 shows experimental results with $\alpha = 60$ and Figure 3.9 shows results with $v = 25$. In both cases, **Towards Visibility Sector** continues to perform significantly better than **Towards Flocking**

Agent in all the configurations utilizing influencing agents and the influencing agent algorithms continue to perform significantly better than the naive method.

Figure 3.9 shows that **Towards Flocking Agent** performs significantly better than **Towards Visibility Sector** for the configurations utilizing the naive method. **Towards Flocking Agent** likely performs better because the slower velocity ($v = 25$ instead of $v = 50$), which means more time steps will often be required, makes moving closer to the flocking agent important for the naive method. This is because moving towards the flocking agent can decrease the number of time steps required for the influencing agent to enter the flocking agents' visibility sector later in the trial.

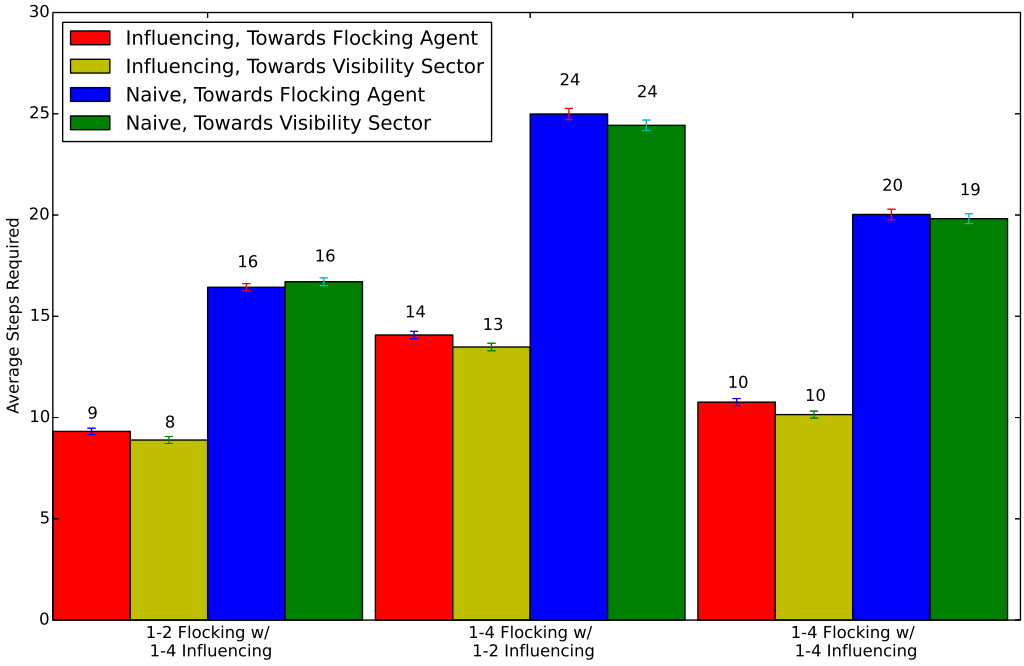


Figure 3.8: Results obtained using FlockSim on three different team configurations over 1000 trials with default settings except that $\alpha = 60$. The error bars depict the standard error of the mean.

3.4 Summary

In this chapter, we considered the problem of leading one or more stationary flocking agents to a desired orientation using influencing agents. This chapter's main contributions were (1) an initial

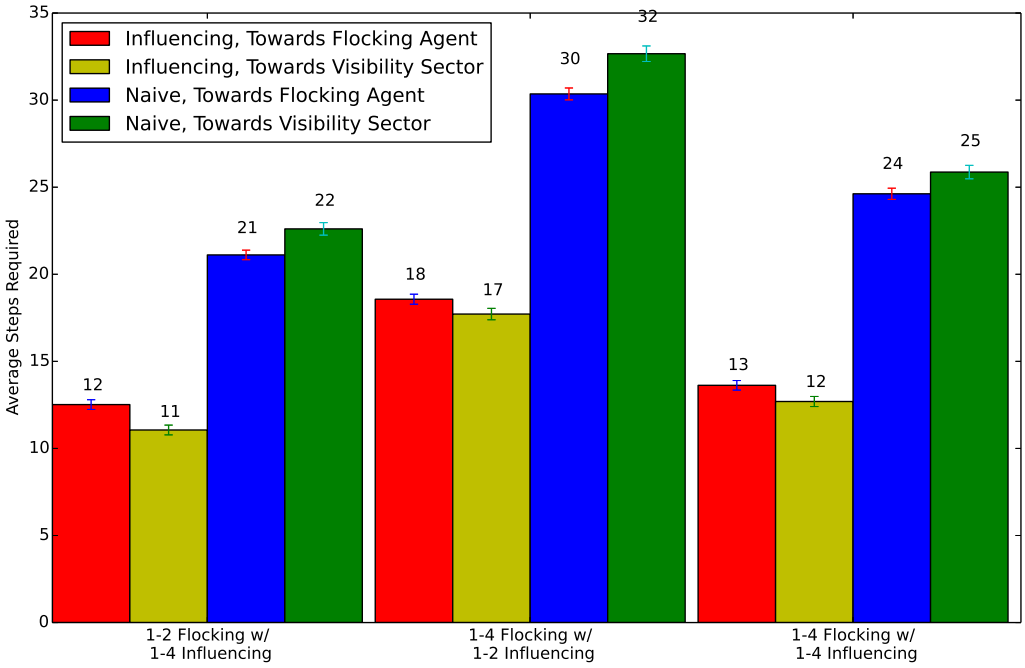


Figure 3.9: Results obtained using FlockSim on three different team configurations over 1000 trials with default settings except that $v = 25$. The error bars depict the standard error of the mean.

theoretical analysis and (2) an empirical analysis of three algorithms for influencing agent behavior. Specifically, in this chapter we set bounds on the extent of influence the ad hoc agents can have on the team when all the agents are stationary and then we subsequently examined the more complicated problem of orienting a stationary team using a set of non-stationary ad hoc agents. Three algorithms for influencing agent behavior were also presented in this chapter. Algorithm 2 — an algorithm for orienting a stationary flock to a desired orientation using a set of non-stationary influencing agents — was analyzed both theoretically and empirically. Algorithms 3 and 4 — which present behaviors for influencing agents that are not within the neighborhoods of any flocking agents — were analyzed empirically due to the complexity of theoretical analysis in this case.

4. Influencing a Flock to a Desired Orientation

In this chapter,[1] we introduce three algorithms that influencing agents can use to influence a flock of agents to adopt a desired orientation θ^*. The previous chapter considered influencing a stationary flock from theoretical and empirical standpoints. This chapter advances past the previous chapter by considering the more general case, where the flocking agents have non-zero velocity and different initial positions.

This chapter addresses two questions: how to *orient* the flock to a target heading and how to *maneuver* a flock through turns. Hence, throughout this chapter we consider two specific cases. In the *Orient* case, the influencing agents attempt to influence the flock to travel towards θ^*. In the *Maneuver* case, the influencing agents attempt to influence the flock to travel as a cohesive unit through multiple turns — this can be thought of as influencing the flock towards a frequently changing θ^*.

Sections 4.1, 4.2, and 4.3 introduce algorithms for determining the heading of each influencing agent at each time step. Specifically, Section 4.1 introduces *1-Step Lookahead*, a 1-step lookahead algorithm. Section 4.2 introduces *2-Step Lookahead*, a 2-step lookahead algorithm. Finally, Section 4.3 introduces *Coordinated*, a 1-step lookahead algorithm where the influencing agents coordinate their behavior. Experimental results for using these three algorithms to determine influencing agent behavior in the *Orient* case are presented in Section 4.4. Section 4.5 considers how the 1-Step Lookahead behavior can be applied to influence a flock to avoid an obstacle in the *Maneuver* case. Videos showing the main contributions from this chapter are available on our web page.[2]

The work in this chapter assumes that each agent utilizes the *visibility radius* neighborhood model described in Section 2.1.1. The *Orient* case utilizes the *Flock Manipulation* performance metric described in Section 2.2.1, while the *Maneuver* case utilizes a hybrid performance metric

[1] This chapter is based on a conference paper [34] and journal article [35] that I wrote with Peter Stone. Author contributions were as follows: I was a Ph.D. student and did the complete implementation and writing. Peter was my advisor — he collaborated with me on deciding research directions and interpreting results.

[2] `http://www.cs.utexas.edu/~katie/videos/`

that borrows elements from both the *Flock Manipulation* performance metric described in Section 2.2.1 and the *Placement* performance metric described in Section 2.2.2. This hybrid performance metric will be described in Section 4.5.1. The challenge of designing influencing agent behaviors in a dynamic flocking system is difficult because the action space is continuous. Hence, in this dissertation we make the simplifying assumption of only considering a limited number (*numAngles*) of discrete angle choices for each influencing agent.

4.1 1-Step Lookahead Behavior

In this section we present the *1-Step Lookahead* influencing agent behavior in Algorithm 5. Algorithm 5 is a greedy, myopic 1-step lookahead algorithm for determining the best individual behavior for each influencing agent, where "best" is defined as the behavior that will exert the most influence on the next time step. This algorithm considers *all* of the influences on neighbors of the influencing agent at a particular point in time, such that the influencing agent can determine the best orientation to adopt based on this information.

Due to the successful evaluation of the *1-Step Lookahead* behavior later in this chapter (Section 4.4), the *1-Step Lookahead* behavior will be utilized as the primary influencing agent behavior from Chapter 5 onward.

The variables used throughout Algorithm 5 are defined in Table 4.1. Two functions are used in Algorithm 5: *neighbor.vel* returns the velocity vector of neighbor and *neighbor.neighbors* returns a set containing the neighbors of neighbor.

Note that Algorithm 5 is called on each influencing agent at each time step, and that the neighbors of the influencing agent at that time step are provided as parameter *neighOfIA* to the algorithm. The output from the algorithm is the orientation that, if adopted by this influencing agent, is predicted to influence its neighbors to face closer to θ^* than any of the other *numAngles* discrete influencing orientations considered.

Conceptually, Algorithm 5 is concerned with how the neighbors of the influencing agent are influenced if the influencing agent adopts a particular orientation at this time step. Algorithm 5 considers each of the *numAngles* discrete influencing agent orientation vectors. For each orientation

Variable	Definition
bestDiff	the smallest difference found so far between the average orientation vectors of *neighOfIA* and θ^*
bestOrient	the vector representing the orientation adopted by the influencing agent to obtain *bestDiff*
iaOrient	the influencing agent orientation vector (one of the (*numAngles*) discrete angle choices)
neighOfIA	the neighbors of the influencing agent
nOrient	the predicted next step orientation vector of neighbor n of the influencing agent if the influencing agent adopts *iaOrient*
nOrients	a set of the predicted next step orientation vectors of all of the neighbors of the influencing agent, assuming the influencing agent adopts *iaOrient*

Table 4.1: Variables used in Algorithm 5.

vector, the algorithm considers how each of the neighbors of the influencing agent will be influenced if the influencing agent adopts that orientation vector (lines 3–13). Hence, Algorithm 5 considers all of the neighbors of each neighbor of the influencing agent (lines 7–11) — if the neighbor of the neighbor of the influencing agent is an influencing agent, the algorithm assumes that it has the same orientation as the influencing agent (even though, in fact, each influencing agent orients itself based on a different set of neighbors, line 9). On the other hand, if it is not an influencing agent, the algorithm calculates its orientation vector based on its current velocity (line 11). Using this information, the algorithm calculates how each neighbor of the influencing agent will be influenced by averaging the orientation vectors of the each neighbor's neighbors (lines 12–13). The algorithm then picks the influencing agent orientation vector that results in the least difference between θ^* and the neighbors' current orientation vectors (lines 14–18).

If there are *numAgents* agents in the flock, the worst-case complexity of Algorithm 5 is calculated as follows. Line 3 executes *numAngles* times, line 5 executes at most *numAgents* times, and line 7 executes at most *numAgents*. Hence, the complexity for Algorithm 5 is O($numAngles * numAgents^2$).

Results regarding how Algorithm 5 performs in terms of the number of time steps needed for the flock to converge to θ^* can be found in Section 4.4.

Algorithm 5 bestOrient = 1StepLookahead(neighOfIA)

```
1:  bestOrient ← (0, 0)
2:  bestDiff ← ∞
3:  for each influencing agent orient vector iaOrient do
4:     nOrients ← ∅
5:     for n ∈ neighOfIA do
6:        nOrient ← (0, 0)
7:        for n' ∈ n.neighbors do
8:           if n' is an influencing agent then
9:              nOrient ← nOrient + iaOrient
10:          else
11:             nOrient ← nOrient + n'.vel
12:       nOrient ← nOrient / |n.neighbors|
13:       nOrients ← {nOrient} ∪ nOrients
14:    diff ← avg diff between vects nOrients and θ*
15:    if diff < bestDiff then
16:       bestDiff ← diff
17:       bestOrient ← iaOrient
18: return bestOrient
```

4.2 2-Step Lookahead Behavior

Whereas the 1-Step Lookahead behavior presented in the previous section optimizes each influencing agent's orientation to best influence its neighbors on the *next* step, it fails to consider more long-term effects. Hence, in this section we present the *2-Step Lookahead* influencing agent behavior in Algorithm 6. Algorithm 6 considers influences on the neighbors of the neighbors of the influencing agent, such that the influencing agent can make a more informed decision when determining the best orientation to adopt.

The variables used in Algorithm 6 that were not used in Algorithm 5 are defined in Table 4.2. Like Algorithm 5, Algorithm 6 is called on each influencing agent at each time step, takes in the neighbors of the influencing agent at each time step, and returns the orientation that, if adopted by this influencing agent, will influence the flock to face closer to θ^* than any of the other *numAngles* influencing orientations considered.

Conceptually, Algorithm 6 is concerned with (1) how the neighbors of each neighbor of the influencing agent are influenced if the influencing agent adopts a particular orientation at this time step (lines 5–13 in Algorithm 6) and (2) how the neighbors of the neighbors of each neighbor of

Algorithm 6 bestOrient = 2StepLookahead(neighOfIA)

```
1: bestOrient ← (0, 0)
2: bestDiff ← ∞
3: for each influencing agent orientation iaOrient do
4:    nOrients ← ∅
5:    for n ∈ neighOfIA do
6:       nOrient ← (0, 0)
7:       for n' ∈ n.neighbors do
8:          if n' is an influencing agent then
9:             nOrient ← nOrient + iaOrient
10:         else
11:            nOrient ← nOrient + n'.vel
12:      nOrient ← nOrient / |n.neighbors|
13:      nOrients ← {nOrient} ∪ nOrients
14:   for each influencing agent orientation iaOrient2 do
15:      nOrients2 ← ∅
16:      for n ∈ neighOfIA do
17:         nOrient2 ← (0, 0)
18:         for n' ∈ n.neighbors do
19:            n'Orient ← (0, 0)
20:            for n" ∈ n'.neighbors do
21:               if n" is an influencing agent then
22:                  n'Orient ← n'Orient + iaOrient
23:               else
24:                  n'Orient ← n'Orient + n".vel
25:            n'Orient ← n'Orient / |n'.neighbors|
26:            if n' is an influencing agent then
27:               nOrient2 ← nOrient2 + iaOrient2
28:            else
29:               nOrient2 ← nOrient2 + n'Orient
30:         nOrient2 ← nOrient2 / |n.neighbors|
31:         nOrients2 ← {nOrient2} ∪ nOrients2
32:      diff ← the avg diff between vects nOrients and θ* and between vects nOrients2 and θ*
33:      if diff < bestDiff then
34:         bestDiff ← diff
35:         bestOrient ← iaOrient
36: return bestOrient
```

Variable	Definition
iaOrient2	the influencing agent orientation vector (one of the (*numAngles*) discrete angle choices) for the second time step
n'Orient	the predicted next step orientation vector of a neighbor *n'* of a neighbor of the influencing agent if the influencing agent adopts *iaOrient*
nOrient2	the predicted "2 steps in the future" orientation vector of neighbor *n* of the influencing agent if the influencing agent adopts *iaOrient* on the first time step and *iaOrient2* on the second time step
nOrients2	a set containing the predicted "2 steps in the future" orientation vectors of all of the neighbors of the influencing agent, assuming the influencing agent adopts *iaOrient* on the first time step and *iaOrient2* on the second time step

Table 4.2: Variables used in Algorithm 6 that were not used in Algorithm 5.

the influencing agent are influenced if the influencing agent adopts a particular orientation at this time step (lines 19–25 in Algorithm 6), since they will influence the neighbors of each neighbor of the influencing agent on the next time step (lines 16–31 in Algorithm 6).

Algorithm 6 starts by considering each of the *numAngles* discrete influencing agent orientation vectors and considering how each of the neighbors of the influencing agent will be influenced if the influencing agent adopts that particular orientation vector. For each neighbor of the influencing agent, this requires considering all of its neighbors and calculating how each neighbor of the influencing agent will be influenced on the first time step (lines 5–13). Next, Algorithm 6 considers the effect of the influencing agent adopting each of the *numAngles* influencing agent orientation vectors on a second time step (lines 14–31). As before, this requires considering all of the neighbors of each neighbor of the influencing agent, and calculating how each neighbor of the influencing agent will be influenced (lines 18–31). However, in order to do this the algorithm must first consider how the neighbors of the neighbors of the influencing agent were influenced by their neighbors on the first time step (lines 20–25). Finally, Algorithm 6 selects the first step influencing agent orientation vector that results in the least difference between θ^* and the neighbors' orientation vectors after both the first and second time steps (lines 32–36).

In Algorithm 6 we make the simplifying assumption that agents do not change neighborhoods within the horizon of our planning. Due to the fact that movements are relatively small with respect to each agent's neighborhood size, the effects of this simplification are negligible for the relatively

small number of future steps that the 2-step lookahead behavior considers.

The complexity of Algorithm 6 can be calculated as follows. Line 3 executes *numAngles* times, line 14 executes at most *numAngles* times, line 16 executes at most *numAgents* times, line 18 executes at most *numAgents* times, and line 20 executes at most *numAgents* times. Hence, the complexity for Algorithm 6 is $O(numAngles^2 * numAgents^3)$.

As with Algorithm 5, results regarding how Algorithm 6 performs in terms of the number of time steps needed for the flock to converge to θ^* can be found in Section 4.4.

4.3 Coordinated Behavior

The influencing agent behaviors presented in Sections 4.1 and 4.2 were for individual influencing agents, where each influencing agent calculated its behavior independent of any other influencing agents. In this section, we present a *Coordinated* influencing agent behavior that considers how influencing agents can coordinate to exert more influence on the flock. In particular, coordination is potentially useful in cases where a flocking agent is in the neighborhoods of multiple influencing agents.

Ideally, all of the influencing agents would coordinate their behaviors to influence the flock to reach θ^* as quickly as possible. However, due to computational considerations, this type of coordinated behavior is infeasible in this work due to the complexity of such a calculation. Instead, we utilize a simplified coordinated behavior. Specifically, we pair influencing agents that share some neighbors. These pairs then work in a coordinated fashion to influence their neighbors to orient towards θ^*. We opted to use pairs for simplicity and for computational considerations, but our approach could also be applied to larger groups of influencing agents that share neighbors.

The *Coordinated* behavior selects the influencing agents to pair by first finding all pairs of influencing agents with one or more neighbors in common. Then a brute-force search finds every possible disjoint combination of these pairs. For each such combination, the sum of the number of shared neighbors across all the pairs is calculated and the combination with the greatest sum of shared neighbors is selected. This combination of chosen pairs is called the *selectedPairs*. Note that *selectedPairs* is recalculated at each time step.

The behavior of each influencing agent depends on whether it is part of a pair in *selectedPairs* or not. If it is part of a pair, it follows Algorithm 7 and coordinates with a partner influencing agent. If it is not part of a pair, it follows Algorithm 5 and performs a 1-step lookahead search for the best individual behavior.

The variables used in Algorithm 7 that were not used in Algorithm 5 or Algorithm 6 are defined in Table 4.3. Only one new function is used in Algorithm 7 that was not used in Algorithm 5 or Algorithm 6. The function is *neighbors.get(x)*, which returns the xth element in the set *neighbors*.

Variable	**Definition**
iaOrientP	the partner's orientation vector (one of the (*numAngles*) discrete angle choices)
nOrientsP	a set used to hold the predicted next step orientation vectors of all the neighbors of the influencing agent's partner, assuming the influencing agent adopts *iaOrient* and the influencing agent's partner adopts *iaOrientP*

Table 4.3: Variables used in Algorithm 7 that were not used in Algorithm 5 or Algorithm 6.

Algorithm 7 is called on one influencing agent in each pair in *selectedPairs* at each time step. Algorithm 7 takes in the neighbors of the influencing agent and the neighbors of the partner of the influencing agent, and returns the orientations that, if adopted by both influencing agents, are guaranteed to influence the flock to face closer to θ^* than any other pair of *numAngles* influencing agent orientations.

Conceptually, Algorithm 7 considers each of the *numAngles* influencing agent orientations for the influencing agent and for the influencing agent's partner and performs two 1-step lookahead searches. The main difference between Algorithm 5 and Algorithm 7 is that the coordinated algorithm takes into account that another influencing agent is also influencing all of the agents that are in both the influencing agent's neighborhood and in the influencing agent's partner's neighborhood. Hence, the influencing agent may choose to behave in a way that influences the other agents in its neighborhood closer to θ^* while relying on its partner to more strongly influence the agents that exist in both of the paired influencing agents' neighborhoods towards θ^*.

Specifically, Algorithm 7 executes as follows. For each potential influencing agent orientation, the algorithm considers how each of the neighbors of the influencing agent will be influenced if

Algorithm 7 bestOrient, bestOrientP = Coordinated(neighOfIA, neighOfP)

```
1: bestOrient ← (0, 0)
2: bestOrientP ← (0, 0)
3: bestDiff ← ∞
4: for each influencing agent orient iaOrient do
5:   for each influencing agent orient iaOrientP do
6:     nOrients ← ∅
7:     for n ∈ neighOfIA do
8:       nOrient ← (0, 0)
9:       for n' ∈ n.neighbors do
10:        if n' is the influencing agent then
11:          nOrient ← nOrient + iaOrient
12:        else if n' is the influencing agent's partner then
13:          nOrient ← nOrient + iaOrientP
14:        else
15:          nOrient ← nOrient + n'.vel
16:      nOrient ← nOrient / |n.neighbors|
17:      nOrients ← {nOrient} ∪ nOrients
18:    nOrientsP ← ∅
19:    for n ∈ neighOfP do
20:      nOrient ← (0, 0)
21:      for n' ∈ n.neighbors do
22:        if n' is the influencing agent then
23:          nOrient ← nOrient + iaOrient
24:        else if n.neighbors.get(n') is influencing agent's partner then
25:          nOrient ← nOrient + iaOrientP
26:        else
27:          nOrient ← nOrient + n'.vel
28:      nOrient ← nOrient / |n.neighbors|
29:      if n ∉ neighOfIA then
30:        nOrientsP ← {nOrient} ∪ nOrientsP
31:    diff ← the avg diff between nOrients and θ* and between nOrientsP and θ*
32:    if diff < bestDiff then
33:      bestDiff ← diff
34:      bestOrient ← iaOrient
35:      bestOrientP ← iaOrientP
36: return bestOrient, bestOrientP
```

the influencing agent adopts that orientation (lines 7–17). Then Algorithm 7 considers how each of the neighbors of the influencing agent's partner will be influenced if the influencing agent's partner adopts each potential influencing agent partner orientation (lines 19–30). Finally, the algorithm selects the influencing agent orientations that result in the least difference between θ^* and the current orientations of the neighbors of both the influencing agent and the influencing agent's partner (lines 31–37). Note that agents that are neighbors of both the influencing agent and its partner are only counted once (lines 29–30).

The complexity of Algorithm 7 can be calculated as follows. Line 4 executes *numAngles* times, line 5 executes *numAngles* times, line 7 executes at most *numAgents* times, line 9 executes at most *numAgents*, line 19 executes at most *numAgents* times, and line 21 executes at most *numAgents*. Hence, the complexity for Algorithm 7 is $O(numAngles^2 * numAgents^2)$.

Results for how Algorithm 7, as well as Algorithms 5 and 6, performed in our experiments can be found in the next section.

4.4 *Orient* Experiments

In this section we describe our *Orient* case experiments. These experiments test the three influencing agent behaviors presented in Sections 4.1, 4.2, and 4.3 against some baseline methods described in this section. Our original hypothesis was that the *1-Step Lookahead* behavior (Algorithm 5), *2-Step Lookahead* behavior Algorithm 6), and *Coordinated* behavior (Algorithm 7) would all perform significantly better with regard to the *Flock Manipulation* performance metric described in Section 2.2.1 than the baseline methods. We also believed that the *2-Step Lookahead* behavior (Algorithm 6) and the *Coordinated* behavior (Algorithm 7) would perform better than the *1-Step Lookahead* behavior (Algorithm 5).

4.4.1 Baseline Influencing Agent Behaviors

In this subsection we describe two behaviors which we use as comparison baselines for the lookahead and coordinated influencing agent behaviors presented in Sections 4.1, 4.2 and 4.3.

Face Desired Orientation Behavior

When following this behavior, the influencing agents always orient towards θ^*. Note that under this behavior the influencing agents do not consider their neighbors or anything about their environment when determining how to behave.

This behavior is modeled after work by Jadbabaie, Lin, and Morse [50]. They show that a flock with a controllable agent will eventually converge to the controllable agent's heading. Hence, the Face Desired Orientation influencing agent behavior is essentially the behavior described in their work, except that in our experiments we include multiple controllable agents facing θ^*.

Offset Momentum Behavior

Under this behavior, each influencing agent calculates the vector sum V of the velocity vectors of its neighbors and then adopts an orientation along the vector V' such that the vector sum of V and V' points towards θ^*. See Figure 4.1 for an example calculation. In Figure 4.1, the velocity vectors of each neighbor are summed in the first line of calculations. In the second line of calculations, the vector sum of the influencing agent's orientation and the results of the first line must equal θ^*, which in this example is pointing directly south. From the equation on the second line of calculations, the new influencing agent orientation vector can be found by vector subtraction. This vector is displayed and then scaled to maintain constant velocity on the third line of calculations.

This behavior was inspired by Algorithm 2 in Chapter 3. However, this behavior (as well as Algorithm 2) fails to consider that the influencing agent is not the only agent influencing its neighbors. In some ways, this behavior could be called a "0-step lookahead" algorithm since it failed to consider any other influences on the influencing agent's neighbors.

4.4.2 Experimental Setup

We utilize the MASON simulator for our experiments in this chapter. The MASON simulator was introduced in Section 2.3.2, but in this section we present the details of the environment that are important for completely understanding the experimental setup utilized for the *Orient* case experiments presented in this section. Figure 2.5(a) shows a sample starting configuration for our

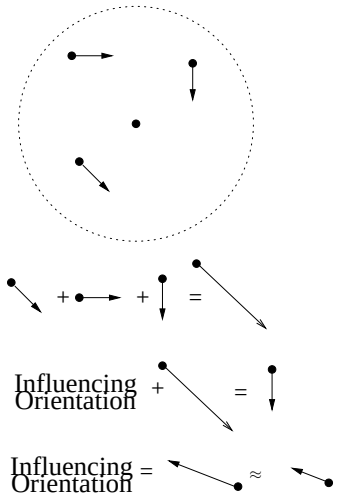


Figure 4.1: An example of how the Offset Momentum influencing agent behavior works. The influencing agent is the black dot, the circle represents the influencing agent's neighborhood, and the three arrows inside the circle represent the influencing agent's neighbors.

Orient case experiments.

We use the default simulator setting of 150 units for the height and width of our experimental domain. Likewise, we use the default setting in which each agent moves 0.7 units during each time step. We also maintained the default toroidal nature of the simulator, such that agents that move off one edge of our domain reappear on the opposite edge moving in the same direction.

The number of agents in our simulation (*numAgents*) is 200, meaning that there are 200 agents in our flock. 10% of the flock, or 20 agents, are influencing agents. The neighborhood for each agent is 20 units in diameter. *numAgents* and the neighborhood size were both default values for MASON. We chose for 10% of the flock to be influencing agents as a trade-off between providing enough influencing agents to influence the flock and keeping the influencing agents few enough to require intelligent behavior in order to influence the flock effectively. Initially, all agents are randomly placed throughout the environment with random initial headings.

We only consider *numAngles* discrete angle choices for each influencing agent. In all of our experiments, *numAngles* is 50, meaning that the unit circle is equally divided into 50 segments beginning at 0 radians and each of these orientations is considered as a possible orientation for each influencing agent. *numAngles*=50 was chosen after some experimentation using the *1-Step Lookahead* behavior in which *numAngles*=20 resulted in a higher average number of steps for the

flock to converge to θ^* and *numAngles*=100 and *numAngles*=150 did not require significantly fewer steps for convergence than *numAngles*=50.

For the *Orient* case, we run 50 trials for each experimental setting. We use the same 50 random seeds to determine the starting positions and orientations of both the flocking agents and influencing agents for each set of experiments for the purpose of variance reduction.

We evaluate performance using the *Flock Manipulation* performance metric described in Section 2.2.1.

4.4.3 Experimental Results

Figure 4.2 shows the number of time steps needed for the flock to converge to θ^* for the two baseline behaviors, the *1-Step Lookahead* behavior presented in Algorithm 5, the *2-Step Lookahead* behavior presented in Algorithm 6, and the *Coordinated* behavior presented in Algorithm 7 using the experimental setup described in Section 4.4.2.

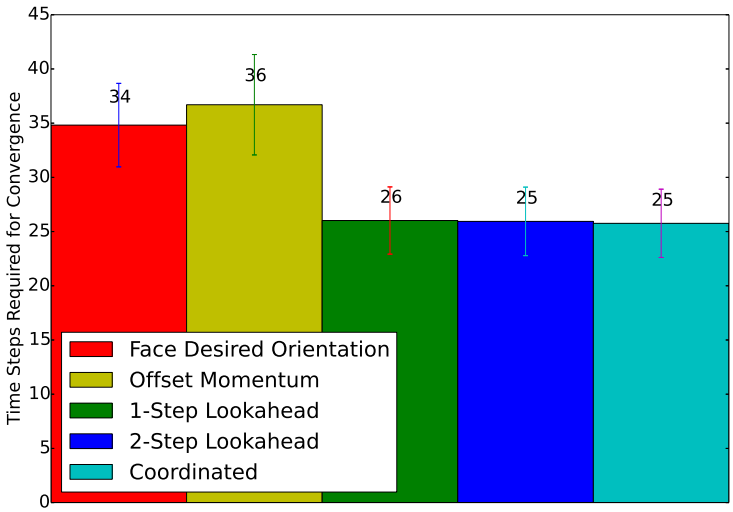


Figure 4.2: Results from *Orient* case experiments using the experimental setup described in Section 4.4.2. The results are averaged over 50 trials and the error bars represent the 95% confidence interval.

Figure 4.2 shows that the *1-Step Lookahead* behavior, the *2-Step Lookahead* behavior, and the *Coordinated* behavior all perform significantly better than the two baseline methods. However,

these results did not show the *2-Step Lookahead* behavior and the *Coordinated* behavior performing significantly better than the *1-Step Lookahead* behavior as we expected. Hence, we present additional experimental results below in which we alter the percentage of the flock that are influencing agents and the number of agents in the flock (*numAgents*) one by one to further investigate the dynamics of this domain.

Altering the Composition of the Flock

Now we consider the effect of decreasing the percentage of influencing agents in the flock to 5% as well as increasing the percentage of influencing agents in the flock to 20%. In both cases, the remainder of the experimental setup is as described in Section 4.4.2. Altering the percentage of influencing agents in the flock clearly alters the number of agents we can control, which affects the how much influence we can exert over the flock. Hence, as can be seen in Figure 4.3, flocks with higher percentages of influencing agents will, on average, converge to θ^* in a lesser number of time steps than flocks with lower percentages of influencing agents.

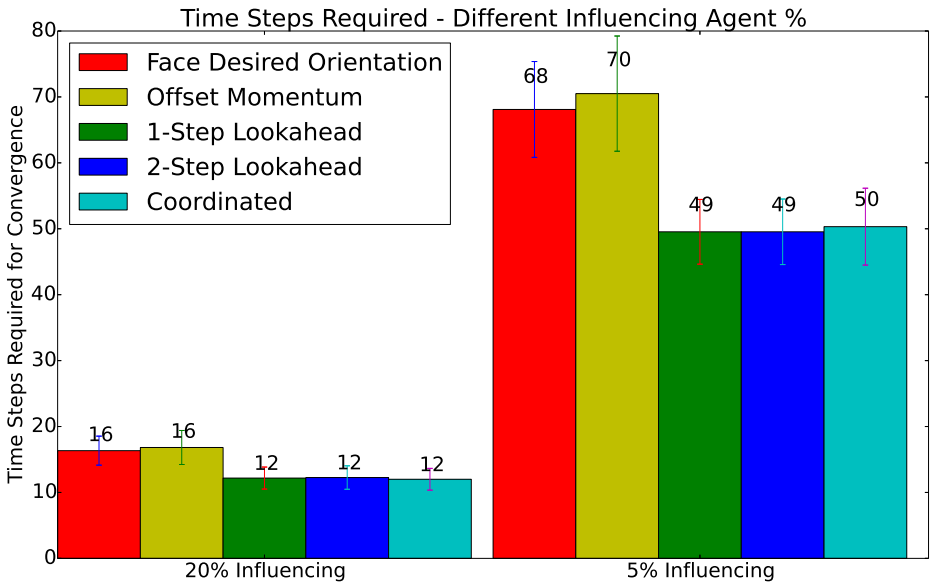


Figure 4.3: Results from *Orient* case experiments using the experimental setup described in Section 4.4.2, except that we varied the percentage of influencing agents in the flock. The results are averaged over 50 trials and the error bars represent the 95% confidence interval.

Altering the Size of the Flock

In this section we evaluate the effect of changing the size of the flock while keeping the rest of the experimental setup as presented in Section 4.4.2. Changing the flock size will alter the number of influencing agents, but not the ratio of influencing agents to non influencing agents. We expected that increasing the flock size would lead to the *Coordinated* behavior performing better comparatively, as with a larger flock, more agents are likely to be in multiple influencing agents' neighborhoods at any given time. However, the *Coordinated* behavior did not perform significantly differently than the lookahead behaviors, and actually performed slightly worse in the experiment with a larger flock size. The results of our experiments in altering the flock size can be seen in Figure 4.4.

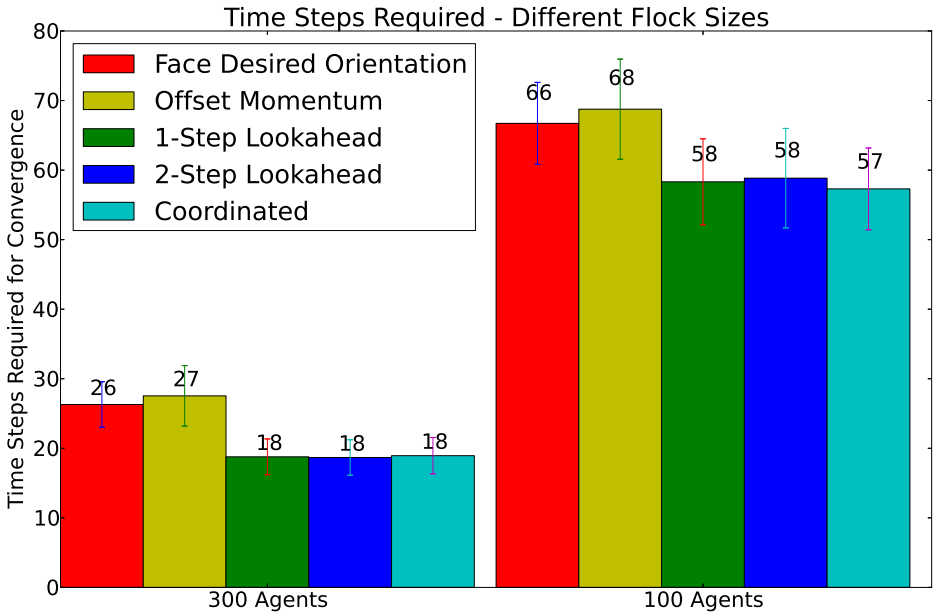


Figure 4.4: Results from *Orient* case experiments using the experimental setup described in Section 4.4.2, except that we varied number of agents in the flock. The results are averaged over 50 trials and the error bars represent the 95% confidence interval.

The difference between the *1-Step Lookahead* behavior, the *2-Step Lookahead* behavior, and the *Coordinated* behavior versus the baseline behaviors was not significant in the experiment utilizing a smaller flock. This may have been caused by the agents being more sparse in the environment,

and hence having less of an effect on each other.

4.4.4 Discussion

Our hypothesis was that Algorithms 5, 6, and 7 would all perform significantly better than the baseline methods. This was indeed the case in all of our experiments except when the flock size was decreased from 200 agents to 100 agents. Apparently having 100 agents in a 150 by 150 unit environment resulted in the agents being too distributed for our lookahead and coordinated behaviors to be effective.

This chapter's original research question, which was to determine how influencing agents should behave so as to orient the rest of the flock towards a target heading as quickly as possible, was partially answered by this work. Although it is possible that better algorithms could be designed, given the algorithms and experimental setting presented in this chapter, we found that it is best for influencing agents to perform the *1-Step Lookahead* behavior presented in Algorithm 5. This behavior is more computationally efficient than the other two algorithms presented, and performed significantly better than the baseline methods in most cases. As such, unless otherwise noted, the *1-Step Lookahead* behavior will be utilized throughout the remainder of this dissertation.

In many cases, the *Coordinated* behavior and the *1-Step Lookahead* behavior led the flock to converge to θ^* in the same number of time steps. This is because the behaviors were identical when no agents were in the neighborhoods of two paired influencing agents at the same time. Additionally, even when a pair of influencing agents shared one or more neighbors, these influencing agents often behaved similarly, and hence did not exert vastly different types of influence.

4.5 *Maneuver* Experiments

So far this chapter has considered the *Orient* case, which studied how influencing agents should orient in order to influence a flock to orient towards a target heading θ^* as quickly as possible. In this section, we consider the *Maneuver* case. Specifically, in this section we consider how the *1-Step Lookahead* behavior can be applied to influence a flock to avoid an obstacle by maneuvering through a set of turns. We opted to use the *1-Step Lookahead* behavior in this section because Section 4.4.4

concluded that — out of the algorithms considered in this chapter — the *1-Step Lookahead* behavior is the best trade-off between computational efficiency and performance.

4.5.1 Experimental Setup

The *Orient* case and *Maneuver* case utilize similar experimental setups. Hence, in this section we only note experimental settings that differ from the *Orient* case experimental setup described in Section 4.4.2. See Figure 2.5 to compare images of sample starting configurations for both the *Orient* case (Figure 2.5(a)) and the *Maneuver* case (Figure 2.5(b)).

In our *Maneuver* case experiments, all of the agents begin within a square in the top left of the domain, where this square occupies 4% of the domain. The agents are initialized within this square with random positions and random headings that are within 90 degrees of the initial θ^*. The influencing agents then influence the flock to travel downward for 300 time steps, then rightward for 300 time steps, then downward for 300 time steps, then leftward for 300 time steps, and finally downward — this path represents the path a flock might need to take to avoid an obstacle in its path.

Different quantities of time steps can be used by the influencing agents to influence the flock to turn during these four turns. The influencing agents alway influence the flock to orient towards θ^*, so during the turns the value of θ^* is interpolated linearly between the values of θ^* on the surrounding straightaways according to the number of time steps allowed for the turn. Hence, θ^* changes more rapidly when fewer time steps are allowed.

Figure 4.5 depicts the approximate path along which the flock is influenced to travel, including a depiction of how turns of different lengths affect this path. Videos of a flock being maneuvered along a similar path are available on our website.[3] We maintain approximately the same time to complete all four turns by shortening the straightaway times depending on the amount of time allocated to turning. Flocks that are influenced by the influencing agents to turn quicker will inherently have the opportunity to finish their last turn quicker (as can be seen in Figure 4.5). Hence, *steps-optimal* represents the minimal number of time steps that could be spent by an agent to complete the four required straightaways and turns.

[3] `http://www.cs.utexas.edu/~katie/videos/`

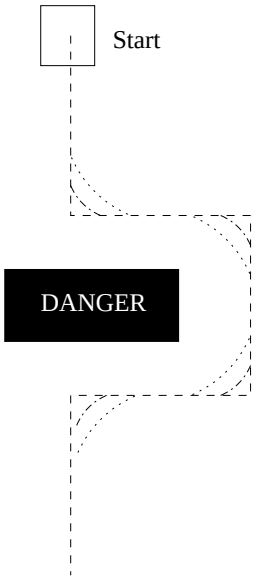


Figure 4.5: The approximate path along which the flock is influenced to travel. The dashed line shows the path if turns were instantaneous and the two arcs show the path when 100 or 200 time steps are used to turn. The flock starts in the square.

For the *Maneuver* case, we increased both the simulator height and width from 150 units to 300 units. We also decreased the units moved by each agent per time step from 0.7 units to 0.2 units. Finally, we altered the simulator to be non-toroidal. Non-toroidal means that agents that move off one edge of our domain become "lost" forever. All of these changes were made to allow the influencing agents additional time and space to "maneuver" the flock.

For the *Maneuver* case, we run 100 trials for each experimental setting. We use the same 100 random seeds to determine the starting positions and orientations of both the flocking agents and influencing agents for each set of experiments.

Our *Maneuver* case experiments utilize a hybrid performance metric that borrows elements from the *Flock Manipulation* performance metric described in Section 2.2.1 and the *Placement* performance metric described in Section 2.2.2. Specifically, for the *Maneuver* case experiments we consider three metrics when determining how much controllability the influencing agents are able to exert on the flock:

1. The average total number of time steps required for the flock to converge to facing downward at the end of the path (*steps-converge*)

2. The difference between *steps-converge* and *steps-optimal* (*diff*)

3. The average number of agents that become separated from the 200-agent flock and do not return to the flock before the flock converges to facing downward at the end of the path (*lost*)

In addition to the three metrics described above, we also report the number of trials in which at least one agent was separated from the flock and did not return before the flock converged to facing downward at the end of the path, as this makes *lost* easier to interpret.

4.5.2 Experimental Results

Table 4.4 shows results of a baseline behavior (Face Desired Orientation Behavior from Section 4.4.1) and the *1-Step Lookahead* behavior (from Section 4.1) using the experimental setup described above for the *Maneuver* case. As can be seen in the table, usage of the *1-Step Lookahead* behavior results in significantly better *steps-converge* and *diff* than the baseline algorithm for each of the turn times tested in the experiment. On average, flocks that are influenced to turn quicker are more likely to have a greater average *diff*. Additionally, note that given this experimental setup, the influencing agents would do best to use around 30 time steps to influence the flock through each turn, as *steps-converge* is least when 30 time steps are used for each turn.

	Steps–Converge	**Steps–Optimal**	**Diff**	**Lost**	**Times Lost**
10 Steps to Turn – Baseline	1243.0 (4.6)	1205	38.0	17.0	1
30 Steps to Turn – Baseline	1242.3 (2.6)	1215	27.3	17.0	1
50 Steps to Turn – Baseline	1245.8 (2.2)	1225	20.8	0	0
100 Steps to Turn – Baseline	1261.0 (1.6)	1250	11.0	17.0	1
200 Steps to Turn – Baseline	1301.9 (1.0)	1300	1.9	17.0	1
10 Steps to Turn – 1-Step Lookahead	1237.0 (5.4)	1205	32.0	13.5	2
30 Steps to Turn – 1-Step Lookahead	1236.5 (4.6)	1215	21.5	17.0	1
50 Steps to Turn – 1-Step Lookahead	1238.6 (3.0)	1225	13.6	17.0	1
100 Steps to Turn – 1-Step Lookahead	1254.5 (1.3)	1250	4.5	0	0
200 Steps to Turn – 1-Step Lookahead	1300.6 (0.6)	1300	0.6	17.0	1

Table 4.4: Results when using the experimental setup described for the *Maneuver* case. The numbers in parentheses show the 95% confidence interval. These results are averaged across 100 trials.

Experiments were run in which the percentage of influencing agents in the flock was altered

to 5% of the flock and 20% of the flock. Results were comparable to those presented in Table 4.4, but did differ in two notable ways. Specifically, when 20% of the flock consisted of influencing agents, no agents were lost during our experiments and turns lasting 10 steps had the least *steps-converge* but were still able to maintain the consistency of the flock. When only 5% of the flock consisted of influencing agents, more influencing agents were lost on quicker turns and turns lasting 50 steps were best in terms of *steps-converge*.

Experiments were also run in which the neighborhood size was decreased. As would be expected, we found that as the neighborhood size becomes smaller, *times lost* increases, *lost* increases, and *steps-converge* increases. Finally, we ran an experiment in which only one of the 200 agents was an influencing agent. Hence, one influencing agent was attempting to influence the entire flock through the series of four turns. In these experiments, we found that a neighborhood of 2000 in diameter was sufficient to not lose any agents on any of our 100 runs.

4.6 Summary

In this chapter, we set out to determine how influencing agents should behave in order to (1) orient a flock towards a target heading as quickly as possible and (2) maneuver a flock around turns quickly while still maintaining the flock. Towards determining how to orient a flock towards a target heading quickly, this chapter introduced three algorithms that the influencing agents can use to influence the flock. Specifically, this chapter introduced a greedy lookahead behavior (*1-Step Lookahead*, Algorithm 5), a deeper lookahead behavior (*2-Step Lookahead*, Algorithm 6), and a coordinated greedy lookahead behavior (*Coordinated*, Algorithm 7). We ran extensive experiments using these algorithms in a simulated flocking domain, where we observed that in such a setting, a greedy lookahead behavior (such as the *1-Step Lookahead* behavior) is an effective behavior for the influencing agents to adopt. This chapter also showed that influencing agents can influence a flock through turns by using the *1-Step Lookahead* behavior and slowly updating the target heading as the flock reaches desired turns. However, we found that the ideal number of time steps for turning depends on the specifics of the domain.

Throughout this chapter, we assumed that influencing agents were initially placed ran-

domly within the flock. Although the *1-Step Lookahead* behavior allowed the influencing agents to effectively influence the flock, we began to wonder if the influencing agents could wield additional influence if they were initially positioned more intentionally. Chapter 5 considers this exact question.

5. Placing Influencing Agents into a Flock

The main contribution of this chapter[1] is a consideration of where to place influencing agents $\{a_0, \ldots, a_{k-1}\}$ into a flock, assuming that once there, they will follow the *1-Step Lookahead* behavior described in Section 4.1's Algorithm 5. In the previous chapter we assumed the influencing agents were added at randomly selected positions to the flock, but in this chapter we assume that we are able to place each influencing agent $a_i \in \{a_0, \ldots, a_{k-1}\}$ into the flock at whatever location $p_i(0)$ we desire at time $t = 0$.

Section 7.1 introduces our experimental settings for this chapter. We introduce these experimental settings before any of our placement methods so that we can present experimental results alongside the methods in each section as well as at the end of the chapter. Section 5.2 introduces three constant-time placement methods and Section 5.3 presents our more effective, but also more computationally expensive, *Graph* placement method. Section 5.4 describes a hybrid method for placing influencing agents into a flock. Specifically, this hybrid method combines the *Graph* placement method from Section 5.3 and the constant-time placement methods from Section 5.2. Section 5.5 introduces a two-step placement method that first identifies many possible influencing agent placement positions and then chooses k of these positions using one of four methods described in this section. Section 5.6 introduces three clustering methods for placing influencing agents into a flock. Finally, Section 5.7 compares the placement methods described in this chapter before Section 5.8 concludes the chapter.

The work in this chapter assumes that each agent utilizes the *visibility radius* neighborhood model described in Section 2.1.1 and the *Placement* performance metric described in Section 2.2.2. Videos showing the placement methods described in this chapter are available on our web page.[2]

[1] This chapter is based on two conference papers [36, 38]. I wrote both with Peter Stone. Shun Zhang also contributed to one of the papers. Author contributions were as follows: I was a Ph.D. student and did the complete implementation and writing. Peter was my advisor — he collaborated with me on deciding research directions and interpreting results. Shun was an undergraduate researcher who contributed work to one paper, but this work was not included in this dissertation.

[2] `http://www.cs.utexas.edu/~katie/videos/`

5.1 Experimental Setup

As in Chapter 4, we use the MASON simulator for our experiments in this chapter. The MASON simulator was introduced in Section 2.3.2, but in this section we present the details of the environment that are important for completely understanding the experimental setup utilized for our placement experiments in this chapter. We discuss our experimental setup at the beginning of this chapter so that experiments can be introduced and discussed throughout the chapter. Figure 2.5(c) shows a sample starting configuration for our placement experiments.

The relevant experimental variables for our placement experiments are given in Table 5.1 .

Variable	**Value**
toroidal domain	no
domain height	300
domain width	300
units moved by each agent per time step (v_i)	0.2
neighborhood for each agent (radius r)	*10*

Table 5.1: Experimental variables for our placement experiments. Italicized values are default settings for the simulator.

Most of our experimental variables in Table 5.1, such as toroidal domain, domain height, domain width, and the units each agent moves per time step, are not set to the default settings for the MASON simulator. We removed the toroidal nature of the domain in order to make the domain more realistic. Hence, if an agent moves off of an edge of our domain, it will not reappear. This is particularly important for *lost* agents remaining *lost*. We also increased the domain height and width, and decreased the units each agent moves per time step, in order to give agents a chance to converge with the flock before leaving the visible area. However, we have no reason to believe the exact experimental settings we chose for our experiments are of particular importance.

All of the experiments reported in this chapter use $m = 10$ or $m = 50$ flocking agents and $k = 2$ to $k = 10$ influencing agents. However, during our initial research, we ran smaller scale experiments with as many as $m = 1000$ flocking agents and $k = 25$ influencing agents. We did not run full scale experiments using all of the methods presented in this chapter for flocks with more than $m = 10$ flocking agents mainly due to the high computation time required for some of the

placement methods. However, our limited experiments did indicate that results from smaller flocks generally do scale to larger flocks.

Figure 2.5(c) shows an example starting configuration for the experiments in this chapter. For the experiments in this chapter, the flocking agents are initially randomly placed within FA_{preset}, which is a small square at the top left of the environment. All of the flocking agents are initially assigned random headings that are within 90 degrees of θ^*.

Experimental results will be presented throughout the following sections and then discussed in Section 5.7. In all of our experiments, we run 100 trials for each experimental setting and we use the same set of 100 random seeds for each set of experiments. The random seeds are used to determine the exact placement and orientation of all of the flocking agents at the start of a simulation experiment. The error bars in all of our graphs depict the standard error of the mean.

5.2 Constant-time Placement Methods

In this section, we consider three simple, constant-time placement methods: *Random* placement in Section 5.2.1, *Grid* placement in Section 5.2.2, and *Border* placement in Section 5.2.3. For each of these placement methods, we consider a *scaled* variant and a *preset* variant.

For the *preset* variant, we assume that the m flocking agents are initially placed within a pre-set area that is formed by the area in which the flocking agents could initially be placed at time $t = 0$ — we refer to this pre-set area as FA_{preset}. Under the *preset* variant, the influencing agent positions are dependent on the space potentially covered by the flocking agents. However, especially for sparse flocks, FA_{preset} is sometimes much larger than the area actually covered by the flocking agents. As such, we also consider a *scaled* variant.

For the *scaled* variant, we scale the area in which the influencing agents are placed based on the area actually occupied by the flocking agents. Specifically, we search all of the locations of the flocking agents and save the highest and lowest x and y values at which flocking agents are located (x_{low}, x_{high}, y_{low}, and y_{high}). We then extend these x and y values by r (the neighborhood radii) and then use the rectangular box formed by $x_{low} - r$, $x_{high} + r$, $y_{low} - r$, and $y_{high} + r$ as the area in which influencing agents can be placed. We call this area FA_{scaled}. Under the *scaled* variant, the

influencing agent positions are dependent on the space actually covered by the flocking agents.

In this section, we discuss both variants for three constant-time placement methods. Section 5.2.1 introduces the *Random* placement method, Section 5.2.2 introduces the *Grid* placement method, and Section 5.2.3 introduces the *Border* placement method. In Section 5.2.4, we compare the three constant-time placement methods and evaluate the variants to determine which should be used with the constant-time placement methods throughout the remainder of this chapter and dissertation.

5.2.1 Random Placement Method

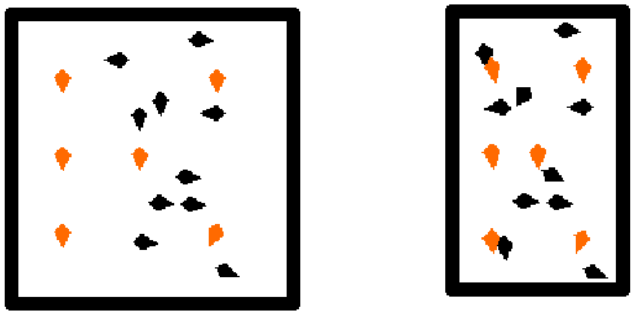
In Chapter 4 we randomly placed k influencing agents within FA_{preset}. Hence, we also use random placement as a baseline method in this chapter. Specifically, the *Random* placement method randomly places the k influencing agents within FA_{preset} under the *preset* variant and within FA_{scaled} under the *scaled* variant.

5.2.2 Grid Placement Method

The *Grid* placement method places k influencing agents at predefined, well-spaced, gridded positions within FA_{preset} under the *preset* variant and within FA_{scaled} under the *scaled* variant. Grids are available that can fit at most x influencing agents, where the smallest grid in which $k \leq x$ is used. Grids are available in which $x \in \{1, 2, 4, 9, 16, 25, 36, \ldots\}$. For each grid size, agents are allocated to the possible positions randomly. Examples of the *Grid* placement method for various values of k can be seen in Figure 5.1.

5.2.3 Border Placement Method

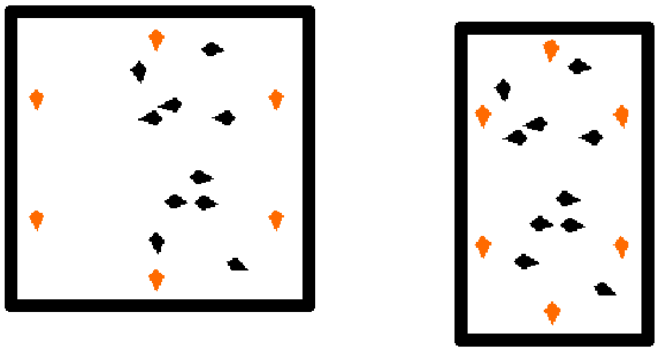
The *Border* placement method places k influencing agents as evenly as possible along the borders of FA_{preset} under the *preset* variant and along the borders of FA_{scaled} under the *scaled* variant. The *Border* placement method places influencing agents on all four sides of the flock until all k influencing agents are placed. At most $\lceil \frac{k}{4} \rceil$ influencing agents are placed on any particular side of the flock. If more than one influencing agent is placed on a particular side of the flock, the



(a) *Grid Preset* (b) *Grid Scaled*

Figure 5.1: Images of influencing agent placement using the *Grid* placement method with *preset* variant (a) and *scaled* variant (b). The dark border shows FA_{preset} in (a) and FA_{scaled} in (b). $m = 10$ and $k = 6$ in both examples. The orange agents are influencing agents while the black agents are flocking agents.

influencing agents spread out as much as possible on that side of the flock. Examples of the *Border* placement method for various values of k can be seen in Figure 5.2.



(a) *Border Preset* (b) *Border Scaled*

Figure 5.2: Images of influencing agent placement using the *Border* placement method with *preset* variant (a) and *scaled* variant (b). The dark border shows FA_{preset} in (a) and FA_{scaled} in (b). $m = 10$ and $k = 6$ in both examples. The orange agents are influencing agents while the black agents are flocking agents.

5.2.4 Experimental Results

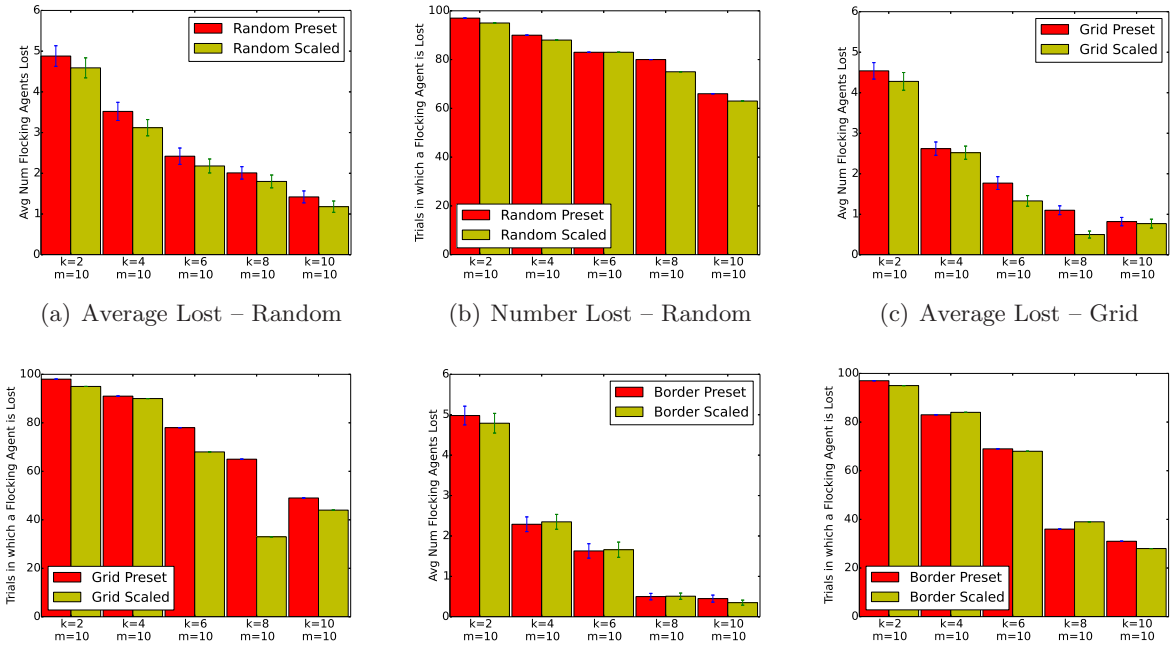
In this section we present experiments that compare the performance of the three constant-time placement methods as well as the *preset* and *scaled* variants for these constant-time placement

methods. The variant that performs best will be utilized with the constant-time placement methods throughout the remainder of this chapter and dissertation.

Comparing *Preset* and *Scaled* Variants

In this section, we compare the two variants in Figures 5.3 and 5.4. Specifically, we compare *Random Preset* to *Random Scaled*, *Grid Preset* to *Grid Scaled*, and *Border Preset* to *Border Scaled*.

Figure 5.3 shows data for a flock with $m = 10$ flocking agents while Figure 5.4 shows data for a flock with $m = 50$ flocking agents. Figures 5.3(a,c,e) and 5.4(a,c,e) show the average number of flocking agents lost and Figures 5.3(b,d,f) and 5.4(b,d,f) shows the total number of trials in which at least one flocking agent is lost. $k = 2$ to $k = 10$ influencing agents are added to the flock in both figures.



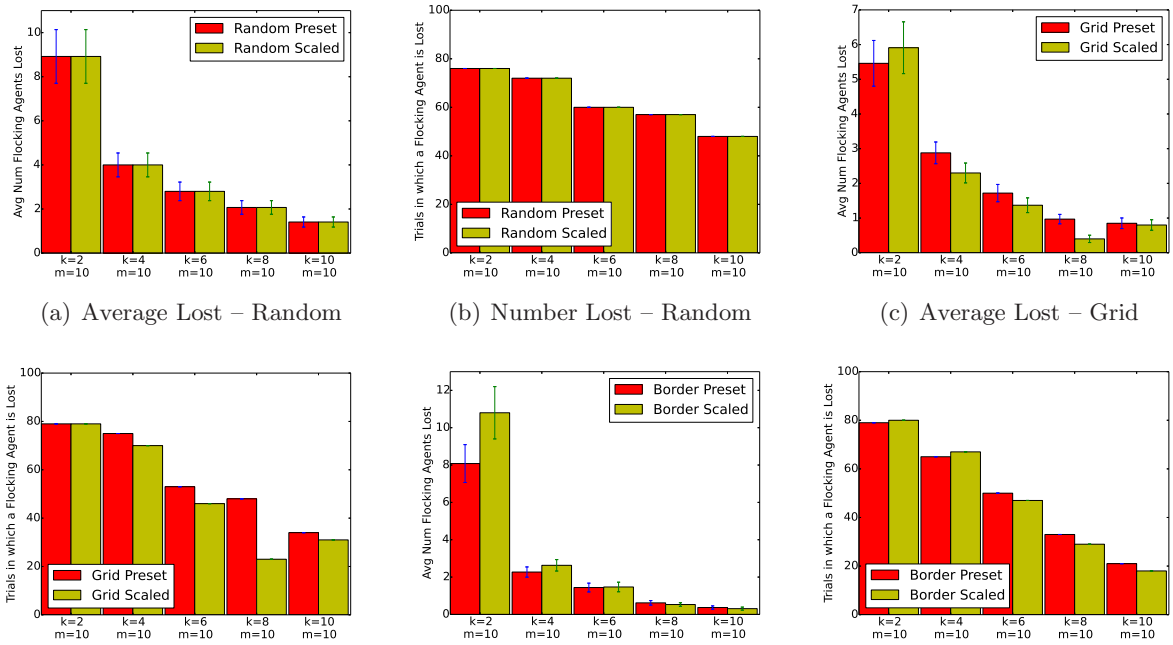
(a) Average Lost – Random (b) Number Lost – Random (c) Average Lost – Grid

(d) Number Lost – Grid (e) Average Lost – Border (f) Number Lost – Border

Figure 5.3: Comparison of *preset* and *scaled* variants for the constant-time placement methods when the flock is comprised of 10 flocking agents ($m = 10$). Graphs (a,c,e) compare the average number of flocking agents lost while graphs (b,d,f) compare the number of trials in which any flocking agents are lost. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

A few interesting trends arise in Figures 5.3 and 5.4. First, across the constant-time place-

ment methods the difference between the *scaled* and *preset* variants is generally not significant — and when the difference is significant, the *scaled* variant almost always loses fewer flocking agents on average than the *preset* variant. One notable outlier is shown in Figures 5.3(e) and 5.4(e). In these figures, *Border Scaled* lost more flocking agents on average than its *preset* counterpart for multiple values of k — but the difference was only significant for $k = 2$ in the $m = 50$ experiments. Second, in terms of the number of trials in which no flocking agents are lost, Figures 5.3(b,d,f) and 5.4(b,d,f) show that the *scaled* variants generally have more trials in which no flocking agents are lost than the *preset* variants. However, there are a few outliers such as $k = 8$ in the $m = 10$ *border* case and $k = 2$ in the $m = 50$ *border* case.



(a) Average Lost – Random (b) Number Lost – Random (c) Average Lost – Grid

(d) Number Lost – Grid (e) Average Lost – Border (f) Number Lost – Border

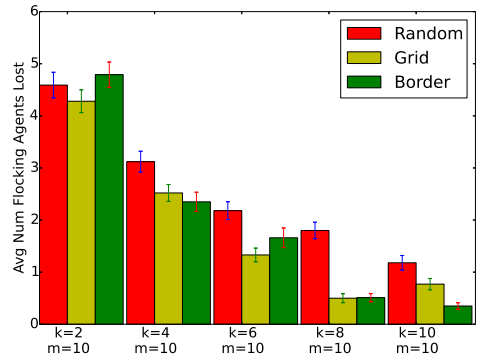
Figure 5.4: Comparison of *preset* and *scaled* variants for the constant-time placement methods when the flock is comprised of 50 flocking agents ($m = 50$). Graphs (a,c,e) compare the average number of flocking agents lost while graphs (b,d,f) compare the number of trials in which any flocking agents are lost. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

Although the difference between the *scaled* and *preset* variants is not significant in most cases — in the cases where the difference is significant, the *scaled* variant almost always performs better. As such, we will utilize the *scaled* variant of each of these constant-time placement methods

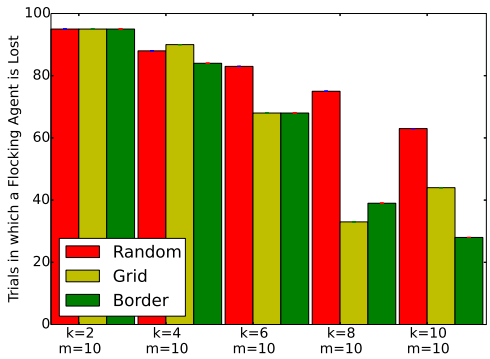
throughout the remainder of this chapter and dissertation. From here forward, each of these methods will be referred to by their shortened names. For example, "*Grid* placement method" will be written instead of "*Grid Scaled* placement method."

Comparing the Constant-time Placement Methods

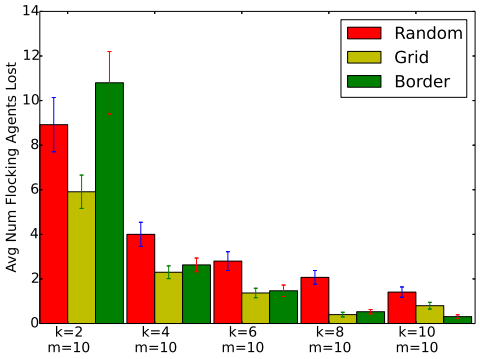
In this section, we compare the three constant-time placement methods in Figure 5.5. Before conducting any experiments, we expected that the *Grid* and *Border* placement methods would perform significantly better than the *Random* placement method. Figures 5.5(a,c) show the average number of flocking agents lost and Figures 5.5(b,d) show the total number of trials in which at least one flocking agent is lost.



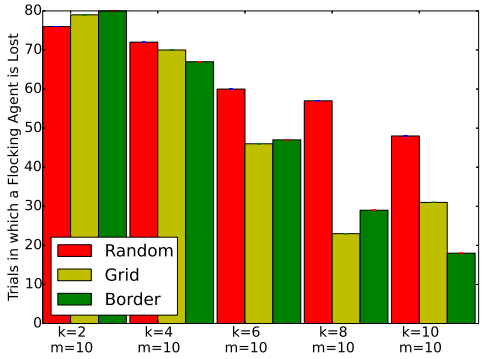
(a) Average Lost ($m = 10$)



(b) Number Lost ($m = 10$)



(c) Average Lost ($m = 50$)



(d) Number Lost ($m = 50$)

Figure 5.5: Comparison of the three constant-time placement methods when k ranges from 2 to 10. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

As can be seen in Figure 5.5(a), the three methods all lose about the same number of flocking agents on average when $k = 2$ and $m = 10$. In Figure 5.5(c), the *Border* placement method performs worse than the *Random* placement method — likely because a "border" cannot be effectively constructed when $k = 2$. However, when $k > 2$, the *Grid* placement method and *Border* placement method both perform significantly better than the *Random* placement method for $m = 10$ and $m = 50$. Additionally, when $k = 10$ the *Border* placement method performs significantly better than the *Grid* placement method. The *Border* placement method likely performs best when $k = 10$ because the two or three influencing agents spread along each border are effectively able to corral any "escaping" flocking agents. Finally, note that the *Grid* placement method performs better when $k = 8$ than when $k = 10$. Generally, we would expect additional influencing agents to improve performance. However, the jump from a 9-agent grid when $k = 8$ to a 16-agent grid when $k = 10$ leads to the influencing agents being unevenly spaced under the *Grid* placement approach when $k = 10$.

5.3 Graph Placement Method

The previous section described various constant-time placement methods. These methods, however, did not consider the positions of each flocking agent when deciding where to place the influencing agents. In this section, we introduce the *Graph* placement method which places influencing agents where they can influence as many flocking agents as possible.

Specifically, the *Graph* placement method considers many possible k-sized sets of positions in which the k influencing agents could be placed, and then evaluates how well each of these sets connects the m flocking agents with the k influencing agents. The set that best connects the m flocking agents with the k influencing agents is chosen. In this section, we describe the *Graph* placement method in detail.

5.3.1 Creating the Graph

All $\{a_k, \ldots, a_{N-1}\}$ flocking agents are added to an initially empty graph G as nodes. Then, for each agent $a_i \in \{a_k, \ldots, a_{N-1}\}$, an undirected edge is added to G between a_i and each of its neighbors

$a_b \in n_i(t)$ if such an edge does not already exist.

5.3.2 Calculating Sets of Influencing Agent Positions

Next the *Graph* placement method considers the positions at which it might add influencing agents. Recall from Section 2.1.1 that r denotes the *visibility radius* of each agent a_i's neighborhood. As indicated in Table 5.1, we assume $r = 10$ in this chapter. For $a_i, a_j \in \{a_k, \ldots, a_{N-1}\}$, the graph placement method considers adding an influencing agent at the mid-point $(\frac{x_i(t)+x_j(t)}{2}, \frac{y_i(t)+y_j(t)}{2})$ between $p_i(t)$ and $p_j(t)$ *only* if $p_i(t)$ and $p_j(t)$ are within $2r$ of each other. This midpoint would allow the agent to influence both a_i and a_j. The graph placement method also considers adding an influencing agent at $(x_i(t) + 0.1, y_i(t) + 0.1)$ for $a_i \in \{a_k, \ldots, a_{N-1}\}$. Note that 0.1 was arbitrarily chosen, but any offset substantially smaller than r would work. This point would allow the agent to at least influence a_i. Our intention was to place the influencing agent exactly in the same place as a_i, but because our environment does not allow two agents to be initially placed in the exact same positions, we instead offset the influencing agent slightly. In cases where no or few flocking agents are within $2r$ of each other, placing influencing agents close to a flocking agent guaranteed that at least one flocking agent would be influenced by each influencing agent.

Once the graph placement method has gathered all of the positions at which it might add influencing agents, it forms all possible k-sized sets of these positions.

5.3.3 Evaluating Sets of Influencing Agent Positions

Finally, the graph placement method takes all of the possible k-sized sets and individually considers each set S of k influencing agent positions. In order to do this, it does the following for each S:

- Add each influencing agent $a_i \in S$ to G
- For each agent $a_i \in S$, an edge is added to G between a_i and each of its neighbors $a_b \in n_i(t)$
- Run the Floyd Warshall shortest paths algorithm on G to obtain the following:
  - **numNoConn**: the number of flocking agents not connected to an influencing agent (directly or indirectly)

- **numConn**: the number of connections between flocking agents and influencing agents (directly or indirectly)
- **numDirectConn**: the number of direct connections between flocking agents and influencing agents
- **numNoDirectConn**: the number of flocking agents not directly connected to an influencing agent

- Remove each influencing agent $a_i \in S$ from G

Once all possible k-sized sets T have been individually considered, the graph placement method selects a set based on the information it obtained. Specifically, it compares in order (lexicographically): (1) minimal **numNoConn**, (2) maximal **numConn**, (3) maximal **numDirectConn**, and (4) minimal **numNoDirectConn**. If only one set matches the description at a level, then it is selected. Otherwise, all of the sets that matched the description at that level are considered at the next level. If multiple sets remain after the final level, one of the remaining sets is chosen randomly.

In practice, we find that a set is usually selected using the first criterion. We have witnessed a few cases in which the fourth criterion has been used, but we have never witnessed a case in which the final criterion of selecting a remaining set randomly has been utilized. Nonetheless, we include it for completeness.

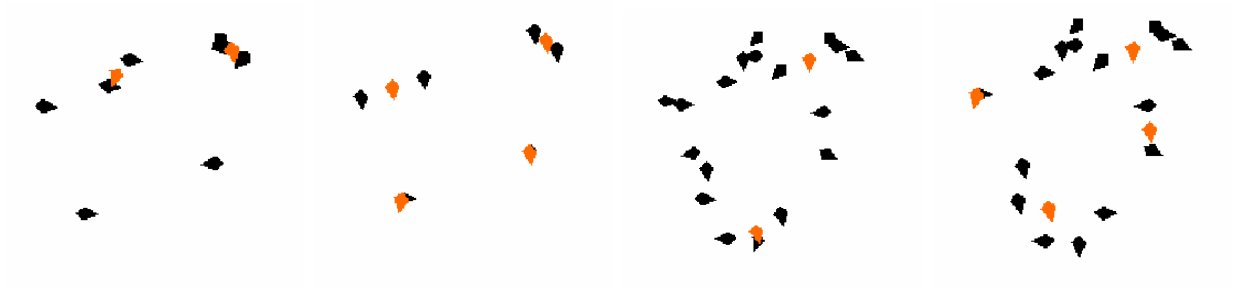
The entire process of selecting placements for k influencing agents, given current placements of m flocking agents, has an algorithmic complexity of $\mathrm{O}(n^3\binom{m^2+m}{k})$.

The Graph Placement Approach can be considered as an instance of the geometric set cover problem. The geometric set cover problem is a special case of the set cover problem in geometric settings. The geometric set cover problem takes in a range space $\sigma = (X, R)$ where X is a universe of points and R is a family of subsets of X called ranges. Ranges are defined by the intersection of X and geometric shapes. The goal is to select a minimum-size subset $C \subseteq R$ of ranges such that every point in the universe X is covered by some range in C. The geometric set cover problem is NP-complete.

Under the Graph Placement Approach, we first define possible influencing agent positions and then we choose k positions at which to place an influencing agent. Using the geometric set cover terminology from the previous paragraph, we describe concretely how the Graph Placement Approach is an instance of the geometric set cover problem. X contains all flocking agent positions, while R is comprised of ranges defined by the indirect influence area of each possible influencing agent position. Specifically, each range is defined as the neighborhoods of the flocking agents directly or indirectly influenced by a possible influencing agent position. We choose at most k ranges in R that best cover the set X of flocking agent positions and then place influencing agents at the influencing agent positions associated with the chosen ranges. This process of choosing at most k influencing agent positions is NP-hard.

There are approximation algorithms for geometric set cover that are polynomial and even near linear time [1, 16]. Approximation is best when R is defined by simple geometric shapes, whereas the geometric shapes defined for the Graph Placement Approach are more complex.

As can be seen in Figure 5.6, our graph placement method places influencing agents such that minimal flocking agents remain uninfluenced. By minimizing the number of flocking agents that are not connected to an influencing agent, our hypothesis was that the *Graph* placement method will be effective at decreasing both the number of lost of flocking agents as well as the number of trials in which any flocking agents are lost.

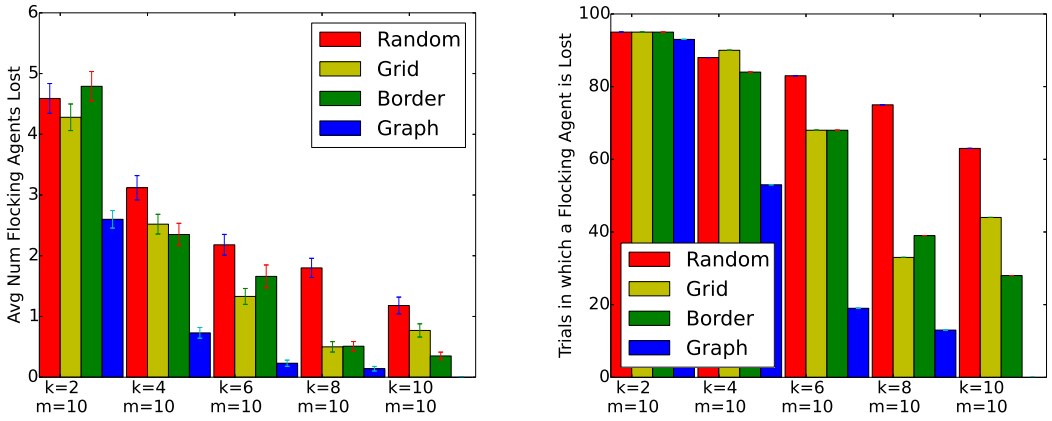


(a) $k = 2$, $m = 8$ (b) $k = 4$, $m = 6$ (c) $k = 2$, $m = 18$ (d) $k = 4$, $m = 16$

Figure 5.6: Images of influencing agent placement using the *Graph* placement method. The orange agents are influencing agents while the black agents are flocking agents.

5.3.4 Experimental Results

In this section, we compare the *Graph* placement method to the three constant-time placement methods from Section 5.2 in Figure 5.7. Our hypothesis before running any experiments was that the *Graph* placement method would perform significantly better than the constant-time placement methods for all values of k tested. Figure 5.7(a) shows the average number of flocking agents lost and Figure 5.7(b) shows the total number of trials in which at least one flocking agent is lost.



(a) Average Lost

(b) Number Lost

Figure 5.7: Comparison of the three constant-time placement methods to the *Graph* placement method when $m = 10$ and k ranges from 2 to 10. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

As can be seen in Figure 5.7, the *Graph* placement method performs significantly better than all three constant-time placement methods for all values of k shown. For $k = 10$, where there are as many influencing agents as flocking agents, the *Graph* placement method does not lose any flocking agents across the 100 trials. This is as expected, since the *Graph* placement method assigns at least one influencing agent to be located within the neighborhood of each flocking agent when $k >= m$.

Note that we were only able to feasibly run complete experiments using $m = 10$ flocking agents due to the $O(n^3\binom{m^2+m}{k})$ computational complexity of the *Graph* placement method. As the $O(n^3\binom{m^2+m}{k})$ computational complexity implies, the computation time grows substantially as both the flock size ($n = k + m$) and the number of flocking agents (m) increases. As discussed in

Section 5.3.3, the Graph Placement Approach can be considered as an instance of the NP-complete geometric set cover problem. As such, in the remaining sections of this chapter we consider quicker heuristic placement methods.

5.4 Hybrid Placement Methods

The *Graph* placement method described in Section 5.3 performed significantly better than the constant-time placement methods described in Section 5.2. However, the *Graph* placement method is not widely and generally useful because the $O(n^3\binom{m^2+m}{k})$ computational complexity limits the sizes of the flocks in which it can be applied.

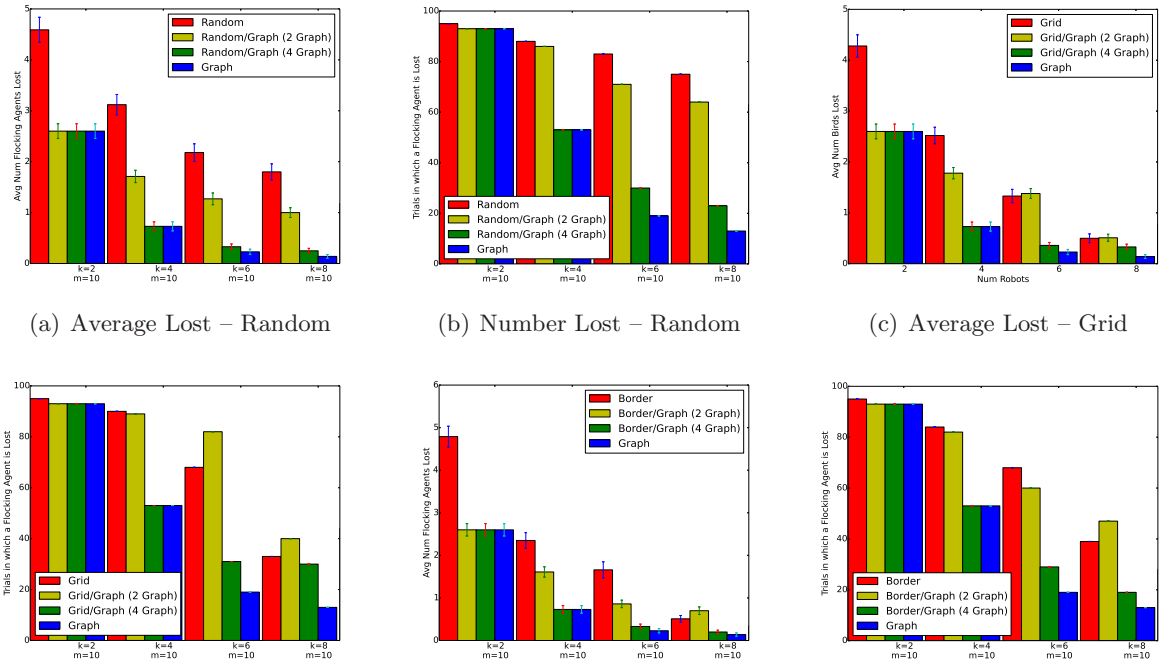
With this computational complexity issue in mind, in this section we consider hybrid methods that utilize the *Graph* placement method to pick the first k_g influencing agent positions and then assign the remaining $k - k_g$ positions based on more computationally efficient methods. The remaining $k - k_g$ positions are randomly chosen from the possible k placements of the more computationally efficient method. In this section, we use the three constant-time placement methods described in Section 5.2 as the computationally efficient methods.

5.4.1 Experimental Results

In this section's experiments, we compare multiple values of k_g as well as multiple placement methods to assign the $k - k_g$ placements not assigned by the *Graph* placement method. Throughout our experiments, the constant-time placement method is either the *Random* placement method from Section 5.2.1, the *Grid* placement method from Section 5.2.2, or the *Border* placement method from Section 5.2.3. The Constant-time/Graph (2 Graph) hybrid placement method places two influencing agents according to the Graph placement method and then places any remaining influencing agents based on the constant-time placement method. Likewise, the Constant-time/Graph (4 Graph) hybrid placement method places four influencing agents according to the Graph placement method and then places any remaining influencing agents based on the constant-time placement method. Results from these experiments are shown in Figure 5.8.

Figures 5.8(a,c,e) show the average number of flocking agents lost when initially placing

influencing agents according to the constant-time, *Graph*, and hybrid placement methods. Likewise, Figures 5.8(b,d,f) show the number of trials in which at least one flocking agent was lost. Before running any experiments, we expected that the *Graph* placement approach would perform best, the Constant-time/Graph (4 Graph) hybrid placement method would perform second best, the Constant-time/Graph (2 Graph) hybrid placement method would perform third best, and the constant-time placement method would perform worst.



(a) Average Lost – Random (b) Number Lost – Random (c) Average Lost – Grid

(d) Number Lost – Grid (e) Average Lost – Border (f) Number Lost – Border

Figure 5.8: Results for the hybrid method experiments when $m = 10$. These graphs compare (a,c,e) the average number of flocking agents lost and (b,d,f) the number of trials in which any flocking agents are lost. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

We strive to minimize both the average number of flocking agents lost and the number of trials in which any flocking agents are lost. The results in Figure 5.8 generally appear as we would expect, but there are a few surprising results. Specifically, Figures 5.8(c,d,e,f) show that for $k = 8$ and $= 10$ both the Grid and Border placement methods lose fewer influencing agents on average — and have fewer trials in which a flocking agent is lost — than the hybrid method that places two influencing agents based upon the *Graph* placement method. This unexpected performance

could be because the hybrid placement method may cover some areas twice while leaving other areas open that would be covered by the Grid and Border placement methods. This hypothesis is supported by the fact that the Random placement graph in Figure 5.8(b) shows that for all k values, the Random placement method loses at least one flocking agent in more trials than the hybrid placement methods.

For all of the graphs in Figure 5.8, for $k = 2$ the results for all of the placement methods except the *Random* placement method are the same — this is expected because both hybrid placement methods use the *Graph* placement method for both influencing agent placements. Likewise, for $k = 4$, the results of the *Graph* placement method and the Constant-time/Graph (4 Graph) method should be the same since both methods use the *Graph* placement method for all four influencing agent placements.

Finally, although the computation complexity is better for the hybrid placement method ($O(n^3\binom{m^2+m}{k_g})$) than for the Graph placement method ($O(n^3\binom{m^2+m}{k})$), the complexity is still dominated by the general flock size. We present run-time comparisons across the various placement methods in Section 5.7.

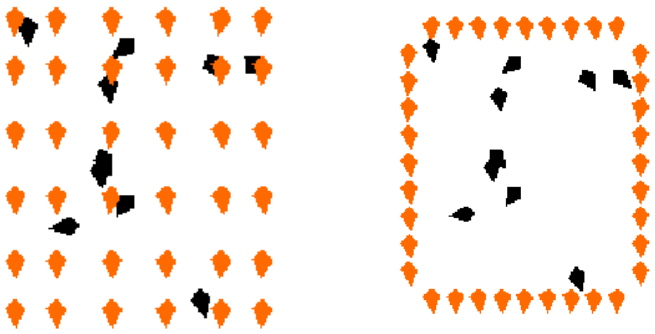
5.5 Two-Step Placement Method

The *Graph* placement method described in Section 5.3 is too computationally intensive to scale to larger flocks and the constant-time placement methods from Section 5.2 are ineffective because the placements were not strongly correlated with the flock's current configuration (henceforth referred to as "flock-aware"). Section 5.4 described a hybrid method for placing influencing agents in a manner that is partially flock-aware and more computationally efficient — but this method was still too computationally intensive. In this section, we present a different method that is partially flock-aware but more computationally efficient.

Specifically, in this section we introduce a two-step method for determining where to initially place influencing agents in a flock. Section 5.5.1 describes Step 1, which is to define a set S of potential influencing agent positions. Section 5.5.2 presents Step 2, which is to select $S' \subseteq S$, where S' contains the k positions at which the influencing agents will be initially placed.

5.5.1 Step 1: Selecting Set S of Possible Influencing Agent Positions

In principle, influencing agents could be placed anywhere. In order to reduce the search space, in this section we define two low complexity methods for defining a set S with many possible influencing agent positions. An example set S for each of these two methods is shown in Figure 5.9.



(a) Grid Set (b) Border Set

Figure 5.9: Example sets of S, where the orange agents are positions in S that could be occupied by the influencing agents. The black agents are flocking agents.

Grid Set

The *Grid Set* method defines a set of gridded positions. Using a small grid reduces the number of positions that Step 2 must consider, while using a larger grid increases the likelihood that each flocking agent will have a potential influencing agent position within its neighborhood. In our experiments, the area in which the flocking agents begin is at most 60 by 60 and the neighborhood radius r is 10 for each agent — due to this, we use a 6 by 6 grid to define 36 potential influencing agent positions. See Figure 5.9(a) for a set of positions defined by the *Grid Set* method.

Border Set

The *Border Set* method defines positions along the boundaries of the area in which the flocking agents begin. The number of positions is a trade-off between the better runtime obtained by limiting the number of positions and the better performance obtained when the influencing agents can be

placed in more effective positions. To maintain consistently with the 36 positions used by the *Grid Set* method, we also use 36 positions for the *Border Set* method. As such, nine potential influencing agent positions are spread evenly along each border. See Figure 5.9(b) for a set of positions defined by the *Border Set* method.

5.5.2 Step 2: Selecting Set $S' \subseteq S$ of k Influencing Agent Positions

Section 5.5.1 described Step 1 of our two-step method for determining how to initially place influencing agents into a flock. In this section, we introduce Step 2. Step 2 selects $S' \subseteq S$, where S' contains the k positions at which the k influencing agents will be initially placed. We introduce four methods for selecting $S' \subseteq S$.

Random

Random randomly selects k positions from S to be in S'. *Random* is included as a baseline since the influencing agents were randomly placed into the *Grid* placement positions and *Border* placement positions in Section 5.2. It is not a completely fair comparison though, since the potential positions in Section 5.2 were closely matched to the number of influencing agents that were being placed whereas in this section we utilize larger sets of *Grid Set* and *Border Set* potential positions.

OneNeighbor

Influencing agents are able to influence flocking agents within their neighborhood. Hence, the simplest flock-aware method for selecting S' given S would be to select influencing agent positions at which the influencing agent has at least one flocking agent as a neighbor. In this section we explain the *OneNeighbor* position selection method that does just this.

Algorithm 8 presents the *OneNeighbor* method. *OneNeighbor* takes in S, a list of current flocking agent positions (*flock*), and the desired number of influencing agent placements (*k'*). The algorithm returns a list containing the *k'* selected initial influencing agent positions.

In all of the algorithms introduced in this chapter, we use consistent notation and terminology. Throughout the following explanation of notation, assume *list* is a list, *item* is an object, and *list2* is another list. *list.add(item)* adds *item* to the end of *list*. *list.size()* returns the number

of objects stored in *list*. *list.removeRandom()* removes a randomly chosen item from *list* while *list.addRandom(list2)* adds a randomly chosen item from *list2* to *list*. The function *areNeighbors(x,y)* takes in two positions, calculates whether agents located at these positions would be neighbors, and returns true or false.

Algorithm 8 OneNeighbor(S,flock,k')

```
 1: positionsNoNeighbors ← {}
 2: positionsWithNeighbors ← {}
 3: for influencingPosition ∈ S do
 4:    numNeighbors ← 0
 5:    for flockingPosition ∈ flock do
 6:       if areNeighbors(influencingPosition, flockingPosition) then
 7:          numNeighbors++
 8:    if numNeighbors > 0 then
 9:       positionsWithNeighbors.add(influencingPosition)
10:    else
11:       positionsNoNeighbors.add(influencingPosition)
12: if k' == positionsWithNeighbors.size() then
13:    return positionsWithNeighbors
14: else if positionsWithNeighbors.size > k' then
15:    while positionsWithNeighbors.size > k' do
16:       positionsWithNeighbors.removeRandom()
17:    end while
18:    return positionsWithNeighbors
19: else
20:    while positionsWithNeighbors.size < k' do
21:       positionsWithNeighbors.addRandom(positionsNoNeighbors)
22:    end while
23:    return positionsWithNeighbors
```

At a high level, *OneNeighbor* begins by calculating the number of flocking agent neighbors each influencing agent would have if placed at each position in S (lines 5–7). Then, k randomly chosen positions that have at least one neighbor are added to S' (lines 12–18). If there are not k positions with at least one neighbor, then some positions with no neighbors are also chosen randomly to be added to S' (lines 19–23). Once S' has k positions, S' is returned.

With this high-level overview in mind, let us walk through Algorithm 8. Lines 3–11 consider each potential initial influencing agent position one by one. For each position, *numNeighbors* counts the number of flocking agents that could be influenced by an influencing agent at this position. If *numNeighbors* is 0 — meaning an influencing agent placed at this position would influence no

flocking agents — then this possible position is added to the *positionsWithNoNeighbors* list (line 11). Otherwise, if *numNeighbors* is greater than 0 — meaning an influencing agent placed at this position would influence at least one flocking agent — then this possible position is added to the *positionsWithNeighbors* list (line 9). Lines 12–13 consider the case where the number of desired placements (*k'*) is equal to the number of influencing agent positions that would influence at least one flocking agent. In this case, *positionsWithNeighbors* is returned. Lines 14–17 consider the case where there are more influencing agent positions that would influence at least one flocking agent than desired placements. In this case, potential placements are randomly removed and discarded from *positionsWithNeighbors* until *positionsWithNeighbors*.size() is equal to the number of desired placements (*k'*). At this point, *positionsWithNeighbors* is returned. Lines 18–21 consider the final case in which the number of desired placements is less than the number of influencing agent positions that would influence at least one flocking agent. In this case, placements are randomly drawn from *positionsWithNoNeighbors* and added to *positionsWithNeighbors* until *positionsWithNeighbors*.size() is equal to the number of desired placements (*k'*). At this point, *positionsWithNeighbors* is returned.

Line 3 executes $|S|$ times (in our experiments, 36 times since S has 36 potential influencing agent positions) and line 5 executes m times. Hence, the algorithmic complexity of Algorithm 8 is $O(|S|^m)$.

MaxNeighbors

The *OneNeighbor* algorithm presented in the previous section calculated the number of flocking agents an influencing agent at each potential influencing position could influence, but only considered whether these values were 0 or greater than zero. The *MaxNeighbors* method presented in this section considers the number of flocking agents each influencing agent could influence and chooses the *k'* influencing agent positions that have the most flocking agents as neighbors.

Algorithm 9 introduces the *MaxNeighbors* method. *MaxNeighbors* takes in the same parameters and returns the same type of information as Algorithm 8.

Most of the notation used in Algorithm 9 was introduced for Algorithm 8. However, we do need to introduce a few new notations for Algorithm 9. Throughout the following explanation of

notation, assume *list* is a list, *item* is an object, *index* is an integer, and *index2* is a different integer great than or equal to *index*. *list.get(index)* returns the item at the position in the list represented by *index*, assuming 0-indexing. *random(index,index2)* returns a random integer greater than or equal to *index* but less than *index2*. *list.remove(index)* removes the item at the position in the list represented by *index*, assuming 0-indexing.

Algorithm 9 MaxNeighbors(S,flock,k')

```
 1: chosenPositions ← {}
 2: numbers ← ()
 3: for influencingPosition ∈ S do
 4:    numNeighbors ← 0
 5:    for flockingPosition ∈ flock do
 6:       if areNeighbors(influencingPosition, flockingPosition) then
 7:          numNeighbors++
 8:    numbers.add(numNeighbors)
 9: for all k' do
10:    maxNeighbors ← 0
11:    maxIndex ← 0
12:    for index = 0 to index = numbers.size() do
13:       if numbers.get(index) ≥ maxNeighbors then
14:          maxNeighbors ← numbers.get(index)
15:          maxIndex ← index
16:    if maxNeighbors == 0 then
17:       maxNeighIndex ← random(0,S.size())
18:    chosenPositions.add(S.get(maxIndex));
19:    numbers.remove(maxIndex);
20:    S.remove(maxIndex);
21: return chosenPositions
```

At a high level, *MaxNeighbors* first calculates the number of flocking agent neighbors an influencing agent would have if placed at each position in S. Then, the k positions in S with the most neighbors are added to S' and S' is returned. With this in mind, let us walk through Algorithm 9. Lines 3–8 consider each potential initial influencing agent position one by one. For each position, *numNeighbors* counts the number of flocking agents that could be influenced by an influencing agent at this position. *numNeighbors* is saved for each potential influencing agent position in *numbers*. Lines 9–20 pick the *k'* influencing agent positions that have the most flocking agent neighbors. On each iteration, lines 10–15 find the maximum value in *numbers* as well as the *index* in *numbers* for this value. Lines 16–17 consider that case where no remaining positions in

S have any neighbors — in this unlikely case, an *index* for *numbers* is chosen randomly. Finally, lines 18–20 add the chosen position to *chosenPositions*, remove the chosen *index* from *numbers*, and remove the chosen position from S. Removing the chosen *index* from *numbers* and the chosen position from S allows for the influencing agent position with the next most neighbors to be chosen on the next iteration.

Line 3 executes $|S|$ times (in our experiments, 36 times since S has 36 potential influencing agent positions) and line 5 executes m times. Hence, the algorithmic complexity of Algorithm 9 is $O(|S|^m)$.

MinUninfluenced

OneNeighbor and *MaxNeighbors* do not consider whether a flocking agent near a potential influencing position is already being influenced by another influencing agent. *MinUninfluenced* addresses this shortcoming by selecting potential influencing agent positions that minimize the number of flocking agents that are "uninfluenced" — or in other words, the number of flocking agents that do not have at least one influencing agent as a neighbor.

Algorithm 10 presents the *MinUninfluenced* method. *MinUninfluenced* takes in the same parameters as Algorithms 8 and 9. It returns a list containing the k' initial influencing agent positions selected by the *MinUninfluenced* method.

Most of the notation used in Algorithm 10 was introduced for Algorithm 8 or 9. However, we do need to introduce a few new notations for Algorithm 10. *list.clone()* returns a deep-copy of list. The function *calculateUninf(flock,positions)* takes in flock and a set of possible influencing agent positions, and then returns the number of uninfluenced flocking agents given the input positions. Likewise, the function *calculateNeigh(flock,positions)* takes in flock and a set of possible influencing agent positions, and then returns the total number of flocking agent neighbors across all influencing agents given the input positions.

MinUninfluenced is a hill climbing algorithm with random restarts. *MinUninfluenced* begins by randomly choosing a set S' of k positions from S (line 5). Then, the algorithm continues as long as an "improvement" is found (lines 15–41). An "improvement" is when substituting any one position in S' with a position from S that is not currently in S' will result in either (1) fewer

Algorithm 10 MinUninfluenced(S,flock,k')

```
1: noImprovement, uninf, neigh ← 0
2: chosenPositions ← ()
3: init ← true
4: while noImprovement < 2 do
5:    chosen ← k' randomly chosen positions from S
6:    unchosen ← all positions in S not in chosen
7:    chosenUninf ← calculateUninf(flock,chosen)
8:    chosenNeigh ← calculateNeigh(flock,chosen)
9:    if init then
10:      init ← false
11:      chosenPositions ← chosen
12:      uninf ← chosenUninf
13:      neigh ← chosenNeigh
14:   improve ← true
15:   while improve do
16:      improve ← false
17:      chosenCount ← 0
18:      while chosenCount < chosen.size() and not improve do
19:         chosenToRemove ← chosen.get(chosenCount)
20:         chosenClone ← chosen.clone()
21:         chosenClone.remove(chosenToRemove)
22:         unchosenCount ← 0
23:         while unchosenCount < unchosen.size() and not improve do
24:            unchosenToAdd ← unchosen.get(unchosenCount)
25:            chosenClone.add(unchosenToAdd)
26:            posUninf ← calculateUninf(flock,chosenClone)
27:            posNeigh ← calculateNeigh(flock,chosenClone)
28:            if posUninf < chosenUninf or (posUninf == chosenUninf and posNeigh > chosenNeigh) then
29:               improve ← true
30:               unchosen.remove(unchosenToAdd)
31:               unchosen.add(chosenToRemove)
32:               chosen ← chosenClone
33:               chosenUninf ← posUninf
34:               chosenNeigh ← posNeigh
35:            else
36:               chosenClone.remove(unchosenToAdd)
37:            unchosenCount++
38:         end while
39:         chosenCount++
40:      end while
41:   end while
42:   if chosenUninf < uninf or (chosenUninf == uninf and chosenNeigh > neigh) then
43:      noImprovement ← 0
44:      uninf ← chosenUninf
45:      neigh ← chosenNeigh
46:      chosenPositions ← chosen
47:   else
48:      noImprovement++
49: end while
50: return chosenPositions
```

uninfluenced flocking agents or (2) the same number of uninfluenced flocking agents but a greater sum of flocking agent neighbors across all influencing agents (line 28). If an improvement is found, then we substitute this influencing agent position (lines 29–34). Once no additional "improvements" can be made, S' is compared to the previous best S' (line 42). If S' has either (1) fewer uninfluenced flocking agents or (2) the same number of uninfluenced flocking agents but a greater sum of flocking agent neighbors across all influencing agents, then S' is saved as the new best S' (lines 42–46). This entire process is repeated until two sequential random restarts have failed to produce a better S'. At this point, the best S' is returned (line 50).

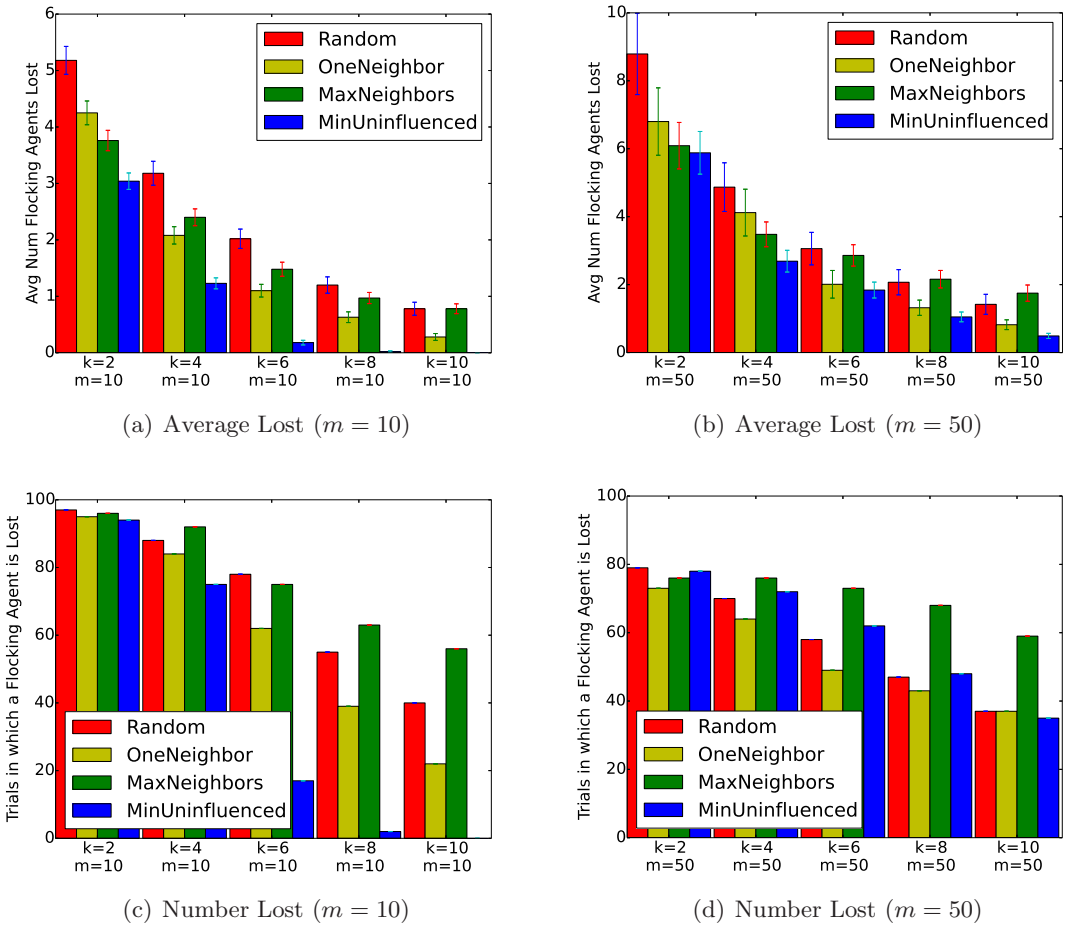
MinUninfluenced finds a locally optimal solution. In principle, it is possible to compute the global optimum by evaluating all subsets of cardinality k. However, evaluating all subsets of cardinality k becomes too computationally complex for greater values of k. As such, we instead utilize the locally optimal hill climbing method with random restarts presented in this section.

5.5.3 Experimental Results

Sections 5.5.1 and 5.5.2 define a two-step method for determining where to initially place influencing agents into a flock. Figures 5.10 and 5.11 show the average number of flocking agents lost (a,b) and the total number of trials in which any flocking agents are lost (c,d) when *Grid Set* (Figure 5.10) and *Border Set* (Figure 5.11) are used to select S and various methods from Section 5.5.2 are used to select S'.

Figure 5.10 shows results for using *Grid Set* to select S. For both $m = 10$ and $m = 50$, using *MinUninfluenced* to select S' loses fewer flocking agents than any other method. For all values of k in Figure 5.10(a) and for $k = 4$ and $k = 10$ in Figure 5.10(b), using *MinUninfluenced* to select S' loses significantly fewer flocking agents than the other methods shown. *MinUninfluenced* performs significantly better because it places a priority on selecting positions that influence flocking agents that are otherwise uninfluenced. Using *OneNeighbor* to select S' does significantly better than *Random* and *MaxNeighbors* for $k > 4$ when $m = 10$ and for $k > 6$ when $m = 50$. *OneNeighbor* performs significantly better because it spreads out influencing agents while *MaxNeighbors* gathers influencing agents in areas with multiple flocking agents.

Figure 5.11 presents results for using *Border Set* to select S. Using *MinUninfluenced* to



(a) Average Lost ($m = 10$)

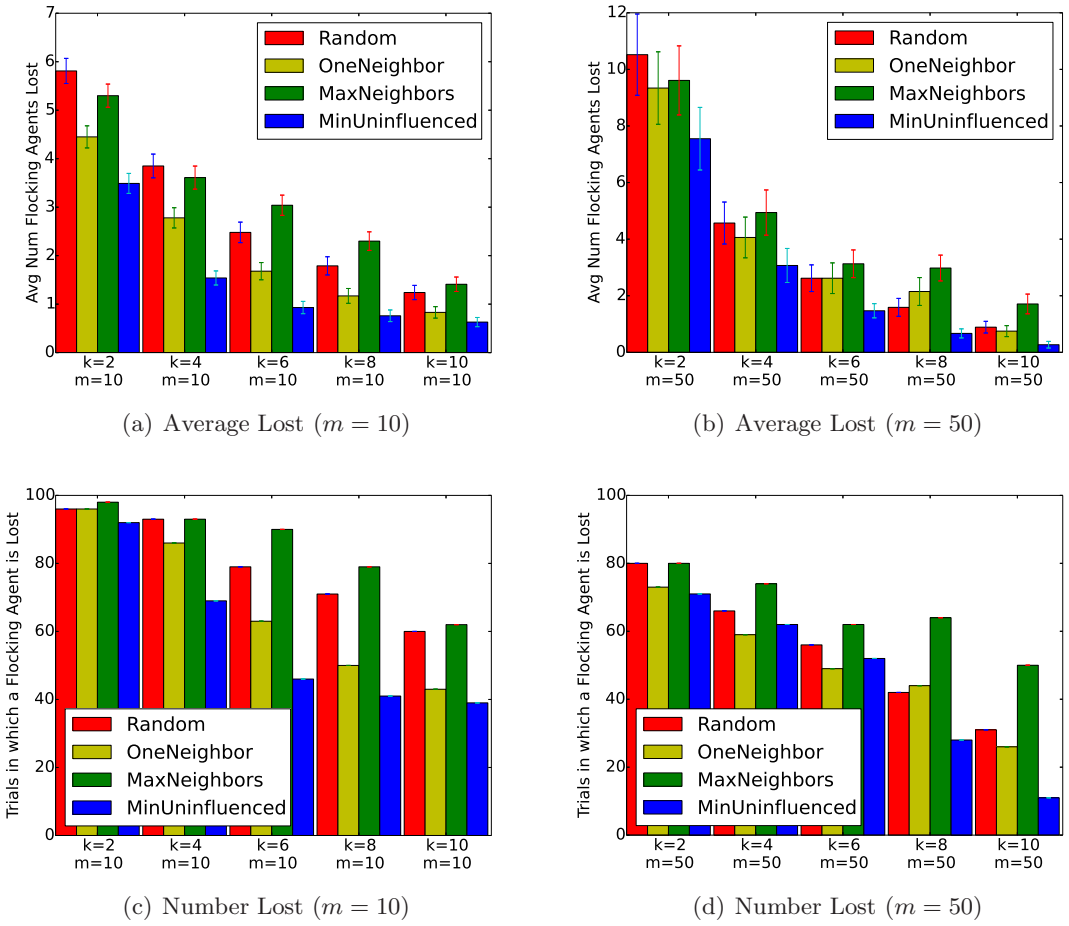
(b) Average Lost ($m = 50$)

(c) Number Lost ($m = 10$)

(d) Number Lost ($m = 50$)

Figure 5.10: Results for when *Grid Set* is used to select S and four methods are used to select S'. These graphs compare (a,b) the average number of flocking agents lost and (c,d) the number of trials in which any flocking agents are lost. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

select S' loses significantly fewer flocking agents on average than the other methods in most cases — specifically for $k < 10$ when $m = 10$ and for $k > 4$ when $m = 50$. *MinUninfluenced* generally performs well because it places influencing agents along the flock border in positions where the flocking agents would not be otherwise influenced. Placing influencing agents along the flock border near flocking agents allows these influencing agents to "save" these flocking agents from leaving the flock and becoming "lost." Figure 5.11(a) shows that using *MaxNeighbors* to select S' performs significantly worse than *OneNeighbor* and *MinUninfluenced*. *MaxNeighbors* performs significantly worse because it clusters influencing agents near groups of flocking agents, while the other methods



(a) Average Lost ($m = 10$)

(b) Average Lost ($m = 50$)

(c) Number Lost ($m = 10$)

(d) Number Lost ($m = 50$)

Figure 5.11: Results for when *Border Set* is used to select S and four methods are used to select S'. These graphs compare (a,b) the average number of flocking agents lost and (c,d) the number of trials in which any flocking agents are lost. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

result in more balanced placements.

5.6 Clustering Placement Methods

The methods presented in Section 5.5 define sets of possible influencing agent positions and then use flock-aware methods to select k positions from these sets. In this section, we use a completely different approach to select initial influencing agent positions from which influencing agents can effectively influence the flock. Specifically, we use three well-recognized clustering methods to identify k clusters of flocking agent positions — Farthest First is described in Section 5.6.1, Expectation

Maximization (EM) is described in Section 5.6.2, and K-Means is described in Section 5.6.3. Once the clusters are identified, we place an influencing agent within each of the k clusters.

Our hypothesis was that using well-recognized clustering methods to identify clusters of flocking agents could be flock-aware yet computationally feasible way to determine effective influencing agent placements. For each clustering method, we use the open-source Weka implementation [41] with default parameters unless otherwise noted. Small-scale experiments indicated that outside of one instance noted below, none of the methods performed significantly better with non-default parameters.

Figure 5.12 shows examples of the various clustering methods presented in this section for $k = 4$ and $m = 10$. Likewise, Figure 5.13 shows examples of the clustering methods for $k = 4$ and $m = 50$.

(a) Farthest First (b) Expectation Maximization (c) K-Means

Figure 5.12: Examples of the clustering methods for $k = 4$ and $m = 10$. The orange agents are influencing agents while the black agents are flocking agents.

5.6.1 Farthest First

Farthest First randomly picks a flocking agent at which to place the first influencing agent. Then, for each subsequent influencing agent placement, the algorithm places an influencing agent at the flocking agent that is farthest from the previously placed influencing agents.

Farthest First guarantees that all of the influencing agents are at least *dist* apart, where *dist* is the distance between the last influencing agent to be placed and the set of the previously placed influencing agents.

The open-source Weka implementation of Farthest First clustering is based upon work by

(a) Farthest First (b) Expectation Maximization (c) K-Means

Figure 5.13: Examples of the clustering methods for $k = 4$ and $m = 50$. The orange agents are influencing agents while the black agents are flocking agents.

Hochbaum and Shmoys [48] as well as by Dasgupta [23]. The specific Weka implementation we use is described in detail on Weka's SourceForge page for the Farthest First class.[3]

5.6.2 Expectation Maximization

Expectation Maximization (EM) is an optimization method that is frequently used for data clustering. EM alternates between performing an expectation (E) step and a maximization (M) step. The Expectation step estimates the probability that each flocking agent position belongs to each cluster. The Maximization step then estimates the parameters of the probability distribution of each cluster. These parameter estimates are then used in the next E step.

Once EM is done iterating between the E and M steps, it assigns a probability distribution to each flocking agent position. This probability distribution indicates the probability of each flocking agent position belonging to each of the k clusters. Although EM could decide how many clusters to create by cross validation, we specified that the algorithm should generate k clusters. We then place an influencing agent at the mean of the normal distribution of each cluster.

The open-source Weka implementation of EM clustering is based upon work by Hartley [44] and Dempster *et al.* [25]. The specific Weka implementation we use is described in detail on Weka's SourceForge page for the EM class.[4]

[3] `http://weka.sourceforge.net/doc.dev/weka/clusterers/FarthestFirst.html`
[4] `http://weka.sourceforge.net/doc.dev/weka/clusterers/EM.html`

5.6.3 K-Means

K-Means chooses k flocking agents as cluster centers. Then, all m flocking agents are assigned to their nearest cluster center and the centroid is calculated for each cluster. These centroids are then set as the new cluster centers. Then, all flocking agents are assigned to their nearest cluster center. This process repeats until convergence. K-Means minimizes the total squared distance between the flocking agent positions and the cluster centers, where the minimum is local.

Small-scale experiments showed that using Farthest First to choose the k flocking agents to be the initial cluster centers did significantly better for most k than choosing the k flocking agents randomly. Hence, we used Farthest First to chose the initial cluster centers.

The open-source Weka implementation of K-Means clustering is based upon work by Arthur and Vassilvitskii [4]. The specific Weka implementation we use is described in detail on Weka's SourceForge page for the K-Means class.[5]

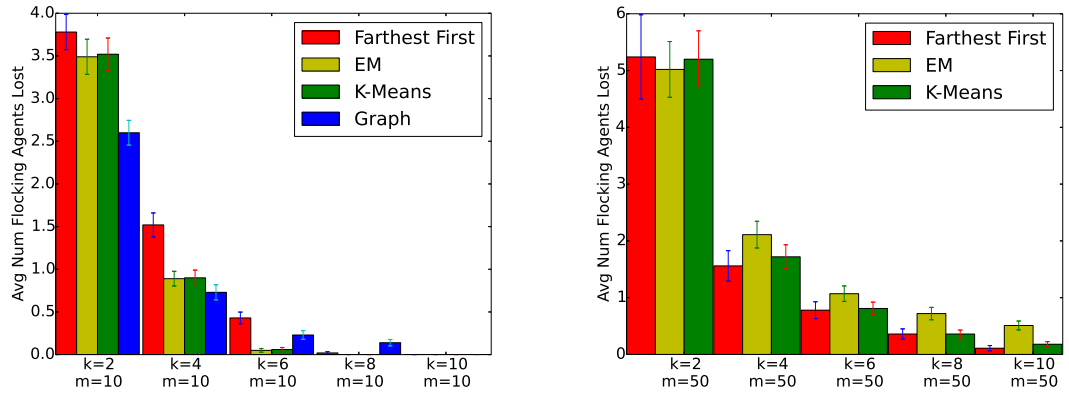
5.6.4 Experimental Results

In this section, we evaluate the clustering methods presented in Sections 5.6.1, 5.6.2, and 5.6.3. We compare these clustering methods to the *Graph placement* method from Section 5.3, since the *Graph* placement method performed best out of the methods discussed in this chapter so far.

Figure 5.14 shows the average number of flocking agents lost when using clustering methods (and the *Graph* placement method for $m = 10$) to select the k initial influencing agent positions. Due to the high complexity and memory usage of the *Graph* placement method, results for the *Graph* placement method are not shown for the $m = 50$ case.

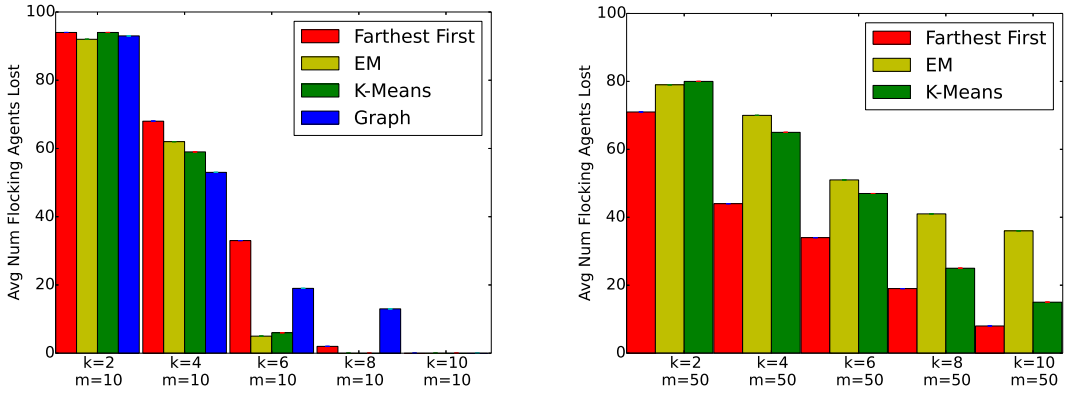
Figures 5.14(a,c) show results for the clustering methods and the *Graph* placement method when $m = 10$. The *Graph* placement method does significantly best for $k = 2$. This is because the *Graph* placement method considers how influence will spread from influencing agents placed at particular positions, which is critical when there are very limited influencing agents. EM and K-Means do significantly best when $k = 6$ and $k = 8$. All four methods perform equally well when $k = m$, as all methods simply assign at least one influencing agent to be in each flocking agent's neighborhood.

[5] `http://weka.sourceforge.net/doc.dev/weka/clusterers/SimpleKMeans.html`



(a) Average Lost ($m = 10$)

(b) Average Lost ($m = 50$)



(c) Number Lost ($m = 10$)

(d) Number Lost ($m = 50$)

Figure 5.14: Comparison of the clustering placement methods (as well as the *Graph* placement method when $m = 10$). These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

Figures 5.14(b,d) show results for the clustering methods when $m = 50$. Farthest First and K-Means do significantly better than EM for $k > 4$. However, only for $k = 10$ does Farthest First perform better than K-Means (although not significantly). Farthest First performs particularly well in the $m = 50$ case because maximizing the distance between the cluster centroids is especially beneficial in a dense flock.

5.7 Discussion

Many different methods for placing influencing agents into a flock were presented in this chapter. Some methods — such as the constant-complexity methods in Section 5.2 — are not flock-aware but scale to large flocks efficiently. Other methods — such as the *Graph* placement method in Section 5.3 — are flock-aware but have computational complexities that do not scale well to larger flocks. In fact, as discussed in Section 5.3.3, the Graph Placement Approach can be considered as an instance of the NP-complete geometric set cover problem. The "sweet spot" lies in methods that are both flock-aware and efficient. In this section, we compare the methods presented in this chapter by considering both their average runtime as well as how to decide which method to use.

5.7.1 Average Runtime

In each section of this chapter, we discussed the computational complexity of each influencing agent placement method. We hypothesized that certain methods would have difficulty scaling to larger flock sizes, so in this section we present runtimes for many of the methods in this chapter. Although we consider runtimes on flocks of various sizes, we use $k = 4$ influencing agents throughout for consistency.

We collected average runtime data for using a variety of methods to select initial influencing agent positions. Experiments were ran on a Dell Latitude E6430 laptop with a 2.9 Ghz CPU. We only report runtimes for the *Border* method because the other constant-time methods have similar runtimes. Likewise, since selecting S using the *Border* and *Grid* methods required approximately the same runtime, we report results for using the *Border* method to select S.

In Table 5.2, *Border (Preset)* and *Border (Scaled)* are baselines — as constant-time placement methods, they represent the minimal feasible runtimes. As would be expected, the *scaled* variant has a slightly greater runtime than the *preset* variant, due to the time required to scale the boundaries. *2-Step (Random)* is generally slightly quicker than *2-Step (OneNeighbor)* and *2-Step (MaxNeighbors)*. *2-Step (MinUninfluenced)* executes slower than the other two-step methods due to multiple random restarts — but it still executes faster than the clustering methods. For the clustering methods, *Farthest First* and *K-Means* scale to larger flock sizes better than EM.

Algorithm	$m = 10$	$m = 50$	$m = 100$
Border (Preset)	8.43 (0.12)	31.72 (0.36)	44.88 (0.49)
Border (Scaled)	8.60 (0.12)	31.99 (0.41)	45.76 (0.45)
Graph	1,076.89 (54.06)	–	–
Hybrid (2 Graph)	185.65 (3.72)	131,865.21 (4,367.33)	–
2-Step (Random)	8.60 (0.12)	31.75 (0.42)	45.30 (0.55)
2-Step (OneNeighbor)	8.97 (0.14)	33.00 (0.34)	45.58 (0.48)
2-Step (MaxNeighbors)	9.59 (0.13)	32.39 (0.36)	45.61 (0.54)
2-Step (MinUninfluenced)	20.77 (0.27)	43.07 (0.44)	55.46 (0.64)
Farthest First	80.10 (2.59)	100.23 (0.64)	121.32 (4.27)
EM	103.28 (1.06)	195.56 (1.38)	266.79 (2.02)
K-Means	108.40 (1.25)	135.01 (0.95)	156.98 (1.68)

Table 5.2: Average run times in milliseconds for each method over 100 trials when $k = 4$. - denotes a configuration for which runtimes could not be obtained due to memory issues related to high computational complexity. Numbers in parenthesis show the standard error of the mean.

As expected, the *Graph* placement method and the hybrid placement method using two graph placements remained too computationally complex to scale. The $m = 50$ experiments for the *Graph* placement method were too computationally intensive, while the hybrid placement method was able to run the $m = 50$ experiments but unable to run the $m = 100$ experiments.

5.7.2 Choosing a Method

Experimental results were presented for each placement method throughout this chapter. In this section we consider when particular methods are preferable. Figure 5.15 provides a flowchart that summarizes the suggestions from this section.

The two-step methods from Section 5.5 usually perform worse — but have better runtimes — than the clustering methods from Section 5.6. Generally, using the *Grid* variant to select S loses fewer flocking agents than using the *Border* variant. However, using the *Border* variant to select S can be better when $m = 50$ and $k > 6$ because then there are enough influencing agents to contain a dense flock within a "border." Using *MinUninfluenced* to select S' usually loses the fewest flocking agents. However, if runtime is a major concern, using *OneNeighbor* to select S' may be preferable.

When $k \leq 4$ and $m = 10$, the Graph placement method from Section 5.3 loses the fewest

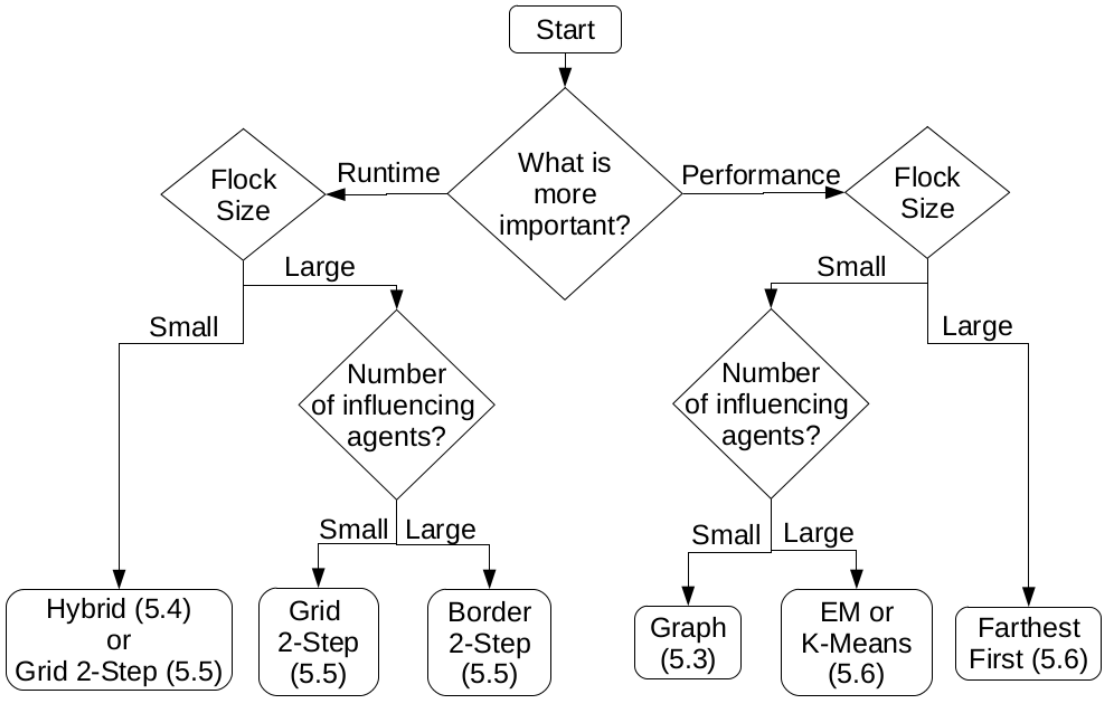


Figure 5.15: A flow chart showing how to select a placement method. Note that different placement methods are preferable in different situations — this flowchart merely makes general suggestions.

flocking agents and has a manageable runtime. Hence, the Graph placement method could be used for small flocks with few influencing agents when runtime is not a concern. When $k > 4$ and $m = 10$, clustering methods EM and K-Means from Section 5.6 lose the fewest flocking agents and have similar run times — as such, EM or K-Means should be used for small flocks with many influencing agents. Farthest First has the best runtime and performance for $m = 50$, so Farthest First should be used for large, dense flocks.

The hybrid placement methods described in Section 5.4 serve as a compromise between the high complexity yet solid performance of the *Graph* placement method from Section 5.3 and the low complexity but weak performance of the constant-time placement methods from Section 5.2. Despite this, using the clustering methods from Section 5.6 is a better choice when the flock is larger. When the flock is smaller, a hybrid placement method would only be preferable over the *Graph* placement method when runtime is more important than performance.

5.8 Summary

In this chapter, we considered how to place influencing agents into a flock such that the influencing agents are able to effectively influence the flock to converge to θ^* while minimizing the number of "lost" flocking agents. In particular, this chapter introduced five types of placement methods for placing influencing agents into a flock. Some of these methods were ineffective because they lost too many flocking agents (i.e. constant-time placement methods from Section 5.2), while others were ineffective because their high computational complexity made them unable to scale to even moderately-sized flocks (i.e. *Graph* placement method from Section 5.3). Other methods fell more moderately along the spectrum of flock-aware yet computationally efficient as can be seen in Figure 5.16.

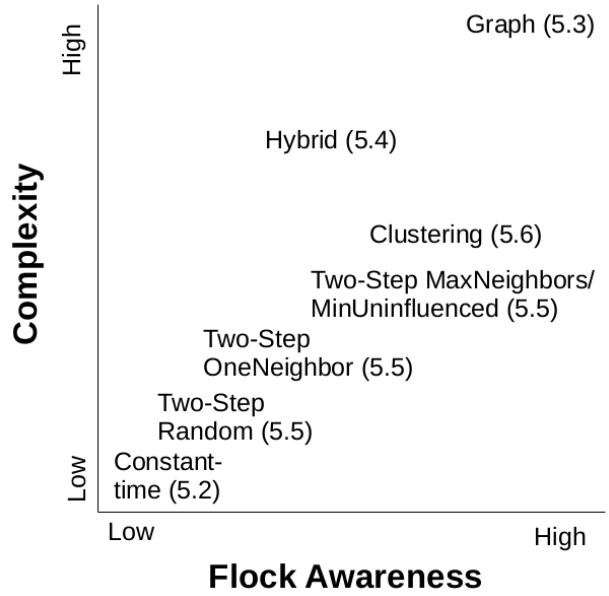


Figure 5.16: Approximate complexity versus flock-awareness of the placement methods discussed in this chapter.

Throughout this chapter, we assumed that influencing agents could be initially placed within a flock. However, this would not be possible if our methods were to be applied to realistic, moving flocks. Chapter 6 considers the more realistic question of how influencing agents could join — and then eventually leave — a flock in motion.

6. Joining and Leaving a Flock

Chapter 4 considered *how* influencing agents should behave in order to influence a flock to alter its flight path. Chapter 5 followed up on the work of Chapter 4 by considering *where* influencing agents should be initially placed into a flock in order to influence the flock. However, both of these chapters assumed that the influencing agents could start *within* the flock. Consider the motivating scenario given in Chapter 1 of using influencing agents to reduce birdstrikes at airports. The assumption that the influencing agents can begin within the flock would not hold if robot birds (acting as influencing agents) were deployed to influence flocks of birds to avoid an airport. In this real-world situation, the robot birds would instead need to leave a charging station near the airport, intercept and join a flock flying towards the airport, influence the flock to alter its current heading, leave the flock without influencing the flock to resume its path towards the airport, and then return to a charging station. While Chapter 4 addresses how the influencing agents can influence the flock to alter its current direction, this chapter addresses the joining and leaving behaviors that would also be necessary.

In this chapter,[1] Section 6.1 presents approaches for joining a flock in motion and Section 6.2 presents approaches for leaving a flock in motion. Section 6.3 describes this chapter's experimental set-up, while Section 6.4 discusses this chapter's experimental results. Section 6.5 concludes the chapter. As such, the research questions addressed by this chapter are: *How should influencing agents join a flock in order to influence it towards a particular behavior? How should influencing agents leave a flock without negatively influencing it?*

The work in this chapter assumes that each agent utilizes the *visibility radius* neighborhood model described in Section 2.1.1 and the *Joining and Leaving* performance metric described in Section 2.2.3.

The flow chart in Figure 6.1 illustrates the process described in this chapter of joining a

[1]Portions of this chapter appear in an extended abstract[37] that I wrote with Peter Stone. Author contributions were as follows: I was a Ph.D. student and did the complete implementation and writing. Peter was my advisor — he collaborated with me on deciding research directions and interpreting results.

flock, influencing the flock, and then leaving the flock. Videos of this process in the MASON simulator are available on our website.[2]

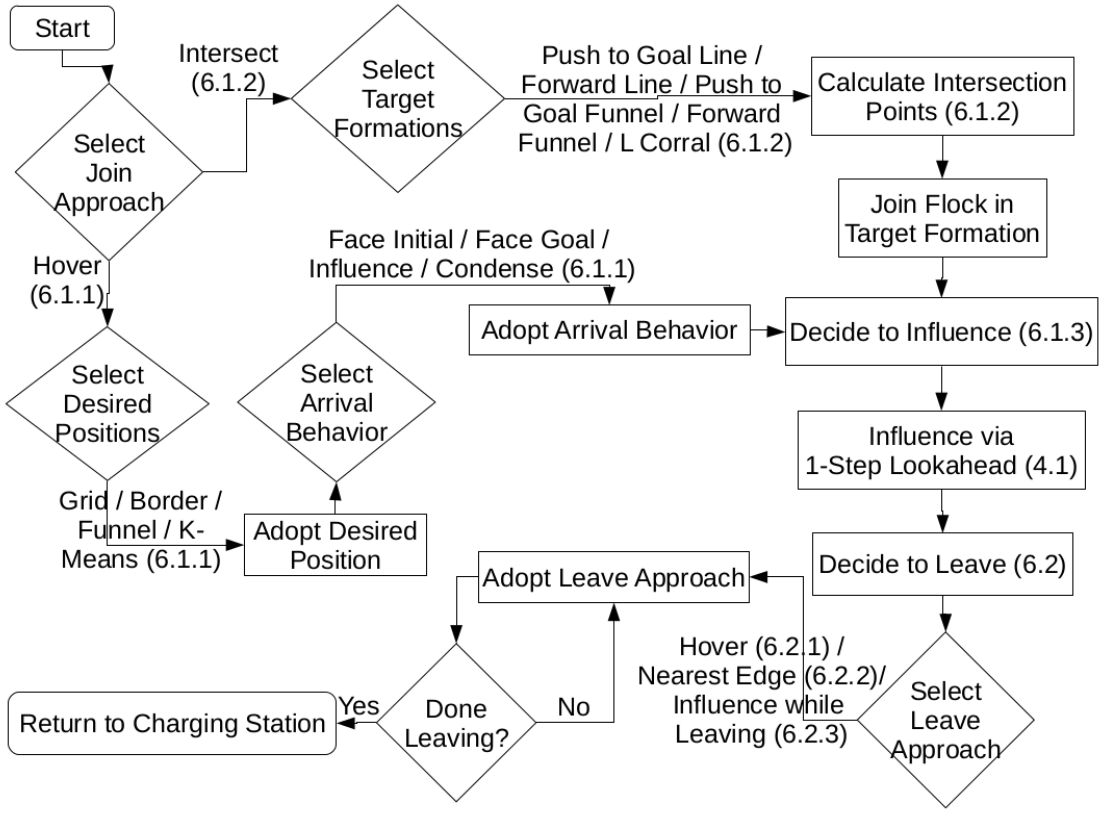


Figure 6.1: This flowchart depicts the procedure described in this chapter of joining a flock, influencing the flock, and then leaving the flock. The numbers in parentheses indicate in which section each part of the procedure is defined. Terminal nodes are represented as ovals, process nodes are represented as rectangles, and decision nodes are represented as diamonds.

6.1 Approaches for Joining a Flock

Chapters 4 and 5, as well as some of the work that will be outlined in Chapter 9, assume that influencing agents can either (1) start within the flock or (2) teleport into the flock. Neither of these assumptions would hold though if robot birds (acting as influencing agents) were deployed to influence flocks in nature. Hence, in this section we consider how influencing agents could join a flock in motion. In particular, we consider two different scenarios for joining — *hover* in which the

[2] `http://www.cs.utexas.edu/~katie/videos/`

influencing agents are able to hover with a particular orientation at a set position (Section 6.1.1) and *intercept* in which the influencing agents leave the charging station at the same velocity as the flocking agents and then maintain this velocity until they return to the charging station (Section 6.1.2).

6.1.1 Hover Approach

The *hover* approach for joining a flock features the influencing agents reaching their desired positions along the flock's flight path ahead of the flock. Once at their desired positions, the influencing agents adopt a particular heading and hover at their desired position. Once the flock is determined to have reached the correct spot with regard to the influencing agents, the influencing agents begin influencing the flock using the 1-Step Lookahead algorithm from Section 4.1.

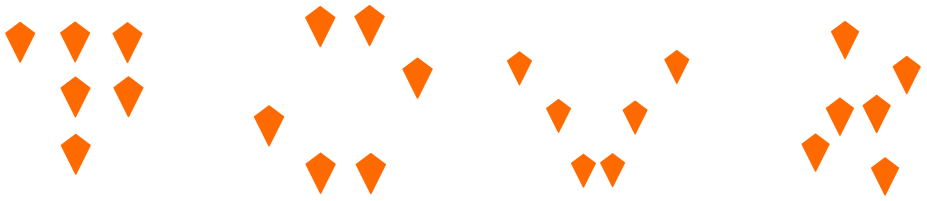
In this section we present multiple methods for (1) selecting desired positions for the influencing agents and (2) determining how the influencing agents should orient before the flock arrives.

Desired Positions

Selecting positions for influencing agents is often a trade-off between effective high computation cost methods and less effective low computation cost methods. Since selection of desired positions needs to occur in real-time and be scalable to large flocks, we consider three successful computationally efficient placement methods — the *Grid* placement method from Section 5.2.2, *Border* placement method from Section 5.2.3, and the *K-Means* method from Section 5.6.3 — and use these methods as position selection methods. These three methods select k positions within the dimensions that will be occupied by the flocking agents once they have reached the correct spot with regard to the influencing agents. We also consider a new *funnel* position selection method, which we describe below. All four position selection methods are depicted in Figure 6.2.

The *funnel* position selection method selects positions in a funnel shape, where the funnel is based on the dimensions that will be occupied by the flocking agents. The base of the funnel is on the side of the flock opposite from where the flocking agents are approaching. There is one influencing agents at the center of the base when k is odd, two influencing agents evenly placed

at the base when k is even, and no influencing agents at the base when $k \leq 2$. The remaining influencing agents are placed evenly along the two sides of the funnel.



(a) Grid (b) Border (c) Funnel (d) K-Means

Figure 6.2: The four *hover* position selection methods for the influencing agents when $k = 6$. Only the exact positions of the influencing agents in (d) are determined by the positions of the flocking agents in the approaching flock.

Arrival Behavior

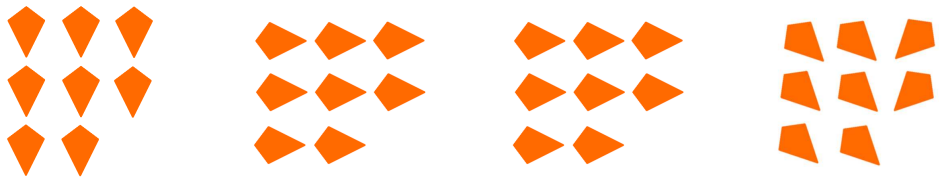
Once the influencing agents are positioned at their desired positions ahead of the approaching flock, their orientation becomes important because the orientation of the influencing agents will influence the flock as it arrives. In this chapter we consider four different arrival behaviors for the influencing agents to adopt while the flock arrives. Examples of each of these arrival behaviors are shown in Figure 6.3.

The *face initial* and *face goal* arrival behaviors behave as would be expected from their names. Influencing agents employing the *face initial* behavior have no influence over an arriving flock, as the flock is already facing the same direction as the influencing agents. On the other hand, the *face goal* behavior begins influencing the flocking agents to orient towards θ^* before the flock has even finished arriving.

The *influence* arrival behavior influences flocking agents towards θ^* using the 1-Step Lookahead algorithm from Section 4.1. We expected this approach would perform similarly to the *face goal* behavior, since both approaches influence the flocking agents towards θ^* before they finish arriving.

Finally, the *condense* arrival behavior orients each influencing agent at a 45° angle towards the mean axis parallel to the flock's initial heading. Although this behavior influences the flocking

agents before they finish arriving, it also condenses the flocking agents. This approach centers around the assumption that a more condensed flock will be easier to influence.



(a) Face Initial (b) Face Goal (c) Influence (d) Condense

Figure 6.3: The four *hover* arrival behaviors when $k = 8$, the *grid* position selection method is used, the flock is approaching from the north, and we want to turn the flock to the east. The orange dots are influencing agents — while the influencing agents in (a), (b), and (d) will not change their orientation as the flock approaches, (c) will actively influence the approaching flock once the influencing agents are within the neighborhood of any flocking agents. Each agent is facing in the direction of its narrow triangular tip — hence, in (a) all of the influencing agents are facing south.

6.1.2 Intercept Approach

The *hover* approach for joining a flock presented in Section 6.1.1 makes one potentially troublesome assumption: that influencing agents that would be seen by the flock as "one of their own" can hover. The makers of the Robird[3] robot bird stated at the 2015 North American Bird Strike Conference that birds believe another bird is "one of its own" if its wing movement and silhouette are the same. This means that winged robot birds may need to be utilized instead of quadcopters or ultralight aircraft. With this in mind, in this section we consider how influencing agents can join a flock without hovering.

Intercept considers the process of joining a flock in motion without hovering. In this section, we present five target formations before discussing our method for calculating when the influencing agents should join the flock. *Intercept* assumes that influencing agents leave from a charging station with the goal of all influencing agents joining and starting to influence the flock at the same time. The influencing agents have the same constant velocity as the flock they are joining. This means that the influencing agents will need to leave the charging station one-by-one, as they will require different amounts of time to reach their target formation desired positions.

[3] `http://clearflightsolutions.com/methods/robirds`

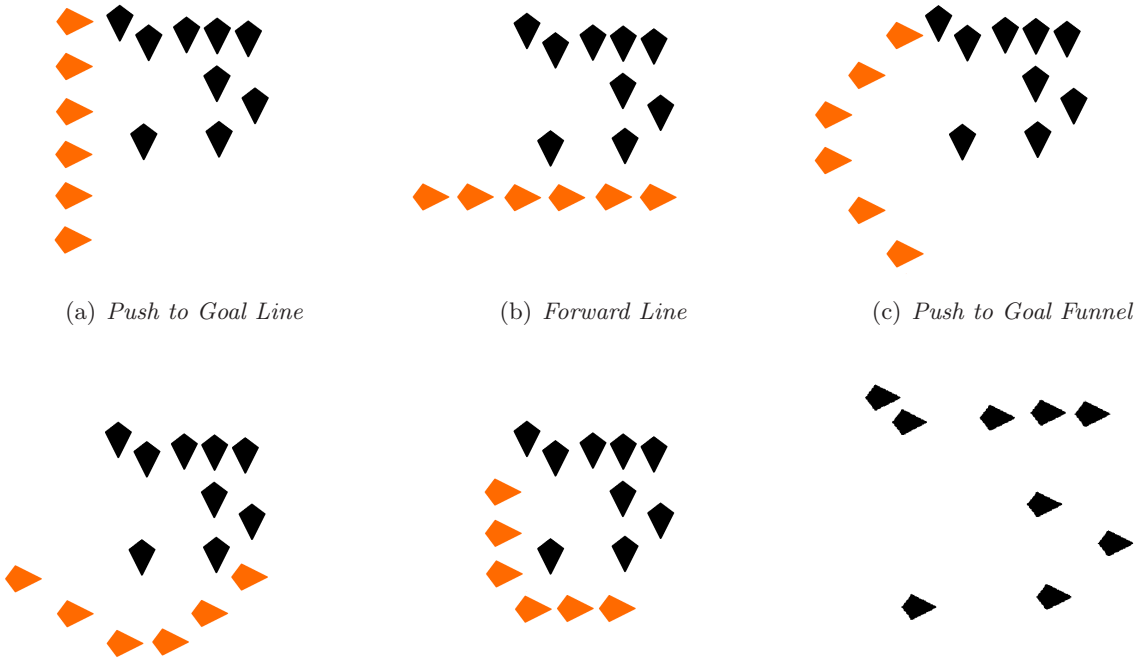
Target Formations

The *hover* approach was able to utilize some placement methods from Chapter 5. The *intercept* approach does not have the same luxury because the placement methods from Chapter 5 did not consider the issue of getting the influencing agents to their desired positions within the flock without influencing the flock while reaching these positions. In fact, using the *Grid* or *K-Means* position selection methods when the influencing agents must travel into the flock to reach their positions results in the flocking agents being influenced to move away from the positioning influencing agents. Hence, in this section we present various target formations that can be used to effectively join a flock.

When determining appropriate target formations, it is important to consider (1) what direction the flock is currently traveling and (2) what direction we instead wish for the flock to travel. As such, we consider five different target formations that are illustrated in Figure 6.4:

1. *Push to Goal Line*: form a line on the side of the flock that is further from the goal initially
2. *Forward Line*: form a line on the side of the flock that reaches areas first as the flock flies initially
3. *Push to Goal Funnel*: form a funnel on the side of the flock that is further from the goal initially
4. *Forward Funnel*: form a funnel on the side of the flock that reaches areas first as the flock flies initially
5. *L corral*: combines the *Push to Goal Line* formation and the *Forward Line* formation

We found that it was generally ineffective to use target formations with target positions on the side of the flock's current location or intended direction. These target formations were ineffective because it was difficult — and sometimes impossible — to get influencing agents in these target formations to be close enough to influence the flock without influencing the flock while reaching their target formation positions.



(a) *Push to Goal Line* (b) *Forward Line* (c) *Push to Goal Funnel*

(d) *Forward Funnel* (e) *L Corral* (f) Desired Orientation of Flock (θ^*)

Figure 6.4: (a)–(e) show the five *intercept* target formations when $k = 6$, the flock is flocking south, and we want to influence the flock to instead travel east. The orange dots are influencing agents while the black dots are flocking agents. In (a)–(e), the flock is shown to be approaching at the exact same time step — this allows the spacial differences between the target formations to be apparent. (f) shows a sample flock facing the desired orientation (θ^*). Each agent is facing in the direction of its narrow triangular tip.

Calculate Interception Points

The target formations described above define where the influencing agents should be with regard to the flock when they intercept the flock. Since the flock is in motion, each influencing agent's desired position according to the target formation moves with the flock at each time step.

For each influencing agent's desired position, there is an exact interception point that can be reached by the flock flocking for x time steps and the influencing agent flying directly towards this interception point for x time steps. For each influencing agent, we find this interception point and the x time steps required to reach it using a binary search algorithm. The greatest x is saved as *joinSteps*, and the point at which the influencing agents will join the flock is set to occur after exactly *joinSteps* time steps.

Each influencing agent then has *joinSteps* to reach its desired position. The exact position

of the flock, and hence of the desired position for each influencing agent, after *joinSteps* can easily be calculated. The influencing agents could all leave at time step 0 if some travelled along curved paths — but for simplicity, we assume that all influencing agents take straight line paths to their desired positions. We calculate how many time steps each influencing agent requires to reach its desired position and schedule it to leave the charging station such that it reaches its desired position after exactly *joinSteps* time steps.

Note that we do not have to assume that the influencing agents take straight line paths. Indeed, curved paths can be used as long as (1) all influencing agents still reach their desired positions at the same time step and (2) no influencing agents prematurely influence the flock on their way to their desired positions.

6.1.3 Decide to Influence

When the flock has arrived, it is time to stop orienting and begin influencing using the 1-Step Lookahead algorithm from Section 4.1. Under the *hover* approach, the flock has arrived when it has reached the point at which the desired positions for the influencing agents were designed. When *face initial* is utilized as the arrival behavior, the desired positions of the influencing agents will align perfectly to the flock. However, other arrival behaviors will result in the actual positions of the influencing agents not aligning exactly to the desired positions of the influencing agents. Inexact alignment is not problematic for our definition of arrival though, as the flock is still said to arrive when it is centered over the area in which the flock was anticipated to intersect the desired positions of the influencing agents. Under the *intercept* approach, the flock has arrived after exactly *joinSteps* time steps.

6.2 Approaches for Leaving a Flock

Once the influencing agents have joined the flock and influenced the flock to face a new orientation using the 1-Step Lookahead algorithm from Section 4.1, they should then leave the flock. To the best of our knowledge, our work is the first to consider how influencing agents should leave a flock after influencing it. Given short battery life on most robots, exiting a flock quickly without

negatively influencing the flock will be important if joining and influencing approaches are to be used with real flocks of birds.

In our work, the influencing agents decide that it is time to transition from influencing the flock to leaving the flock once all of the flocking agents are facing within 5° of θ^*. In this section, we present three approaches for leaving a flock: the *hover* approach in Section 6.2.1, the *nearest edge* approach in Section 6.2.2, and the *influence while leaving* approach in Section 6.2.3.

6.2.1 Hover Approach

The *hover* approach for leaving a flock is similar to the *hover* approach for joining a flock. In this case, all of the influencing agents hover in place facing θ^* when it is time to leave.

The main drawback of the *hover* approach for leaving is that it requires the influencing agents to hover in place. As discussed in Section 6.1, hovering in place could be problematic if robot birds (acting as influencing agents) that would be recognized by birds as "one of their own" are mechanically unable to hover due to their design. For this reason, in the next two sections we consider leaving approaches that do not require the influencing agents to hover.

6.2.2 Nearest Edge Approach

The *nearest edge* approach for leaving a flock requires each influencing agent to know its approximate position within the flock. When it is time to leave, each influencing agent orients towards the nearest edge of the flock that is not the edge facing θ^*. The influencing agents do not try to exit the flock towards θ^* because this would mean they would remain in the flock as the flock continues traveling towards θ^*.

This chapter uses the *Joining and Leaving* performance metric described in Section 2.2.3. The *Joining and Leaving* metric considers the number of birds that intersected an airport in the original direction of travel. Hence, in our experiments we consider two types of the *nearest edge* approach for leaving. The first type is the *nearest 3-edge* approach, which functions exactly as described above. The second type is the *nearest 2-edge* approach, which does not allow influencing agents to exit the flock towards the airport.

6.2.3 Influence while Leaving Approach

One problem with the *nearest edge* approach is that sometimes flocking agents will follow the influencing agents when they leave the flock. In this section, we present the *influence while leaving* approach which leaves the flock more intentionally. A flowchart describing this approach is provided in Figure 6.5; a description is given below.

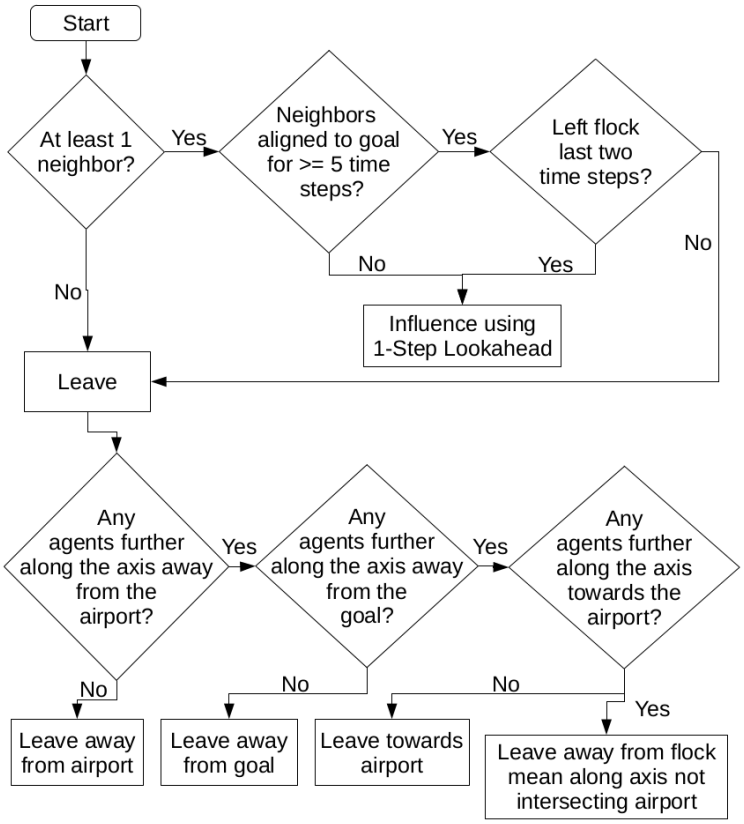


Figure 6.5: This flowchart depicts the decision process at each time steps for the *influence while leaving* approach. Terminal nodes are represented as ovals, process nodes are represented as rectangles, and decision nodes are represented as diamonds.

The *influence while leaving* approach for leaving a flock constantly trades off between influencing the flock towards θ^* and leaving the flock. Specifically, influencing agents utilizing the *influence while leaving* approach behave according to the 1-Step Lookahead algorithm from Section 4.1 to influence the flock towards θ^* unless particular criteria are met. Specifically, an influencing agent behaves to influence its neighbors while it has at least one bird as a neighbor and either the influencing agent has tried to leave the flock for two consecutive time steps or the neighbors of

the influencing agent have been aligned towards θ^* for less than five time steps. Otherwise, the influencing agent can attempt to leave the flock for at most two consecutive time steps. When determining how to exit the flock, each influencing agent considers whether any non-goal direction has no flocking agents further along its axis. If multiple directions have no flocking agents, preference is given to leaving in the direction opposite the airport, followed by the direction opposite the goal — and towards the airport as a last resort. If all directions have some flocking agents, then the influencing agent leaves along the axis not intersecting the airport in the direction away from the flocking agents' mean location.

6.3 Experimental Setup

As in Chapters 4 and 5, we use the MASON simulator for our experiments in this chapter. The MASON simulator was introduced in Section 2.3.2, but in this section we present the details of the environment that are important for completely understanding the experimental setup utilized for our joining and leaving experiments in this chapter. Figure 2.5(d) shows a sample starting configuration for our placement experiments.

The relevant experimental variables for our joining and leaving experiments are given in Table 6.1 .

Variable	**Value**
toroidal domain	no
domain height	600
domain width	600
units moved by each agent per time step (v_i)	0.2
neighborhood for each agent (radius)	*10*

Table 6.1: Experimental variables for our joining and leaving experiments. Italicized values are default settings for the simulator.

As in Chapter 5, most of our experimental variables in Table 6.1 are not set to the default settings for the MASON simulator. We used the same non-toroidal domain, neighborhood, and the units each agent moves per time step as in Chapter 5. However, for the experiments in this chapter we doubled both the domain height and the domain width — increasing the domain size

by a factor of four. This larger domain gives the influencing agents enough space to join the flock, influence the flock, and then leave the flock before the flock leaves the domain.

All of the experiments reported in this chapter use $m = 10$ flocking agents and $k = 2$ to $k = 10$ influencing agents. We did run smaller scale experiments during our initial research with $m = 100$ flocking agents and $k = 25$ influencing agents. However, we decided to use smaller flocks because we found that larger flocks were easier to influence. The larger flocks were easier to influence because the flocking agents were closer to each other and hence had more neighbors. Similarly, we decided to use fewer influencing agents because our limited experiments showed that leaving the flock was more time consuming and had a higher likelihood of negatively influencing the flock as more influencing agents joined the flock.

Figure 2.5(d) shows an example starting configuration for the experiments in this chapter. For the experiments in this chapter, the flocking agents begin within a small 60 by 60 square in the top middle of the environment. All flocking agents begin facing directly south towards the 120 by 60 airport located in the bottom middle of the environment. These flocking agents behave according to the flocking model described in Section 2.1.2. The influencing agents join the flock using approaches presented in Section 6.1 and leave the flock using approaches presented in Section 6.2. In between joining and leaving, the influencing agents influence the flock to orient towards θ^* using the 1-Step Lookahead algorithm from Section 4.1.

Experimental results are presented in Section 6.4. In all of our experiments, we run 100 trials for each experimental setting and we use the same set of 100 random seeds for each set of experiments. The random seeds are used to determine the exact placement of all of the flocking agents at the start of a simulation experiment. The error bars in all of our graphs depict the standard error of the mean.

6.4 Experimental Results

In Sections 6.1 and 6.2 we presented various approaches by which influencing agents can join and then leave a flock. In this section, we evaluate these approaches.

In Section 6.4.1 we present results for the case where the influencing agents can hover in

place. Although this may be feasible for some types of influencing agents, there may also be types of influencing agents (such as fixed-wing robots) that may not be able to hover. Hence, in Section 6.4.2 we also consider the case in which hovering is not possible and instead the influencing agents travel at a constant velocity.

6.4.1 Hovering Experiments

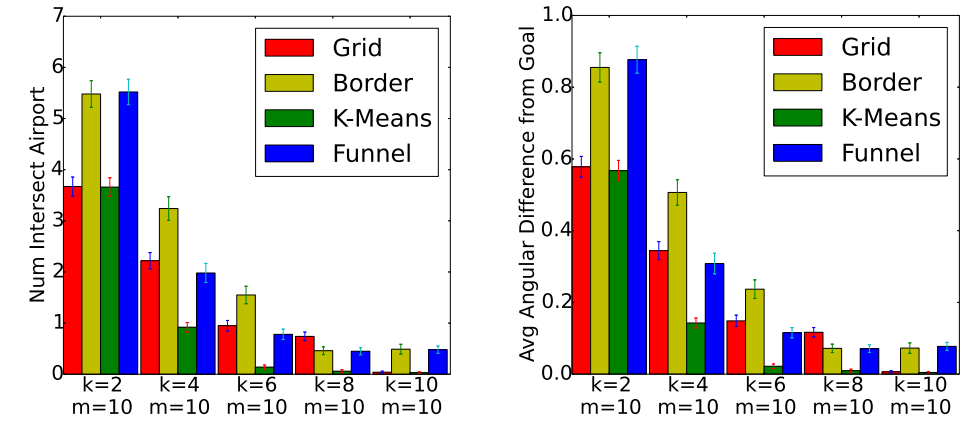
In the case where the influencing agents are able to hover in place while maintaining a particular heading, the influencing agents will use the *hover* approach for both joining and leaving the flock. Following Section 6.1.1, the two main questions that can be asked are: *Which position selection methods perform best? When are various arrival behaviors best?*

First, consider the various position selection methods discussed in Section 6.1.1. Figure 6.6 shows results for the position selection methods using the *face initial* arrival behavior. Across all four metrics, the *K-Means* approach followed by the *Grid* approach performed best. The *K-Means* approach performed best because it selected positions based on the formation of the oncoming flock. Likewise, the *Grid* approach performed well because it ensured that the influencing agents were well-spaced throughout the flock.

Next, consider the four arrival behaviors presented in Section 6.1.1. Figure 6.7(a) shows that *face goal* and *influence* consistently intersect less flocking agents with the airport than *face initial*. As shown in Figure 6.3, this is because *face goal* and *influence* direct the flocking agents towards θ^* as the flock is arriving. This early influence away from the airport leads to these approaches having significantly less flocking agents intersect with the airport. However, this early influence also leads these approaches to have significantly fewer flocking agents orienting within 10° of θ^* (as seen in Figure 6.7(d)).

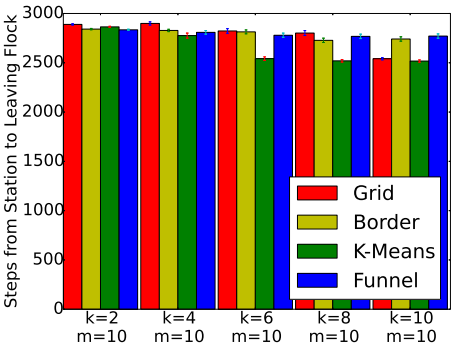
6.4.2 Intercept Experiments

In Section 6.4.1 we considered the situation in which the influencing agents can hover while joining and leaving the flock. Since some influencing agents — such as fixed wing robots — may be unable to hover, in this section we consider the situation in which the influencing agents maintain a constant velocity. Specifically, there are two main questions that should be asked: *Which target*

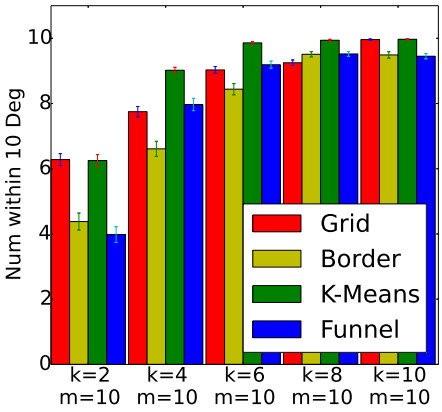


(a) Num Intersect Airport

(b) Avg Diff from θ^*



(c) Time Steps



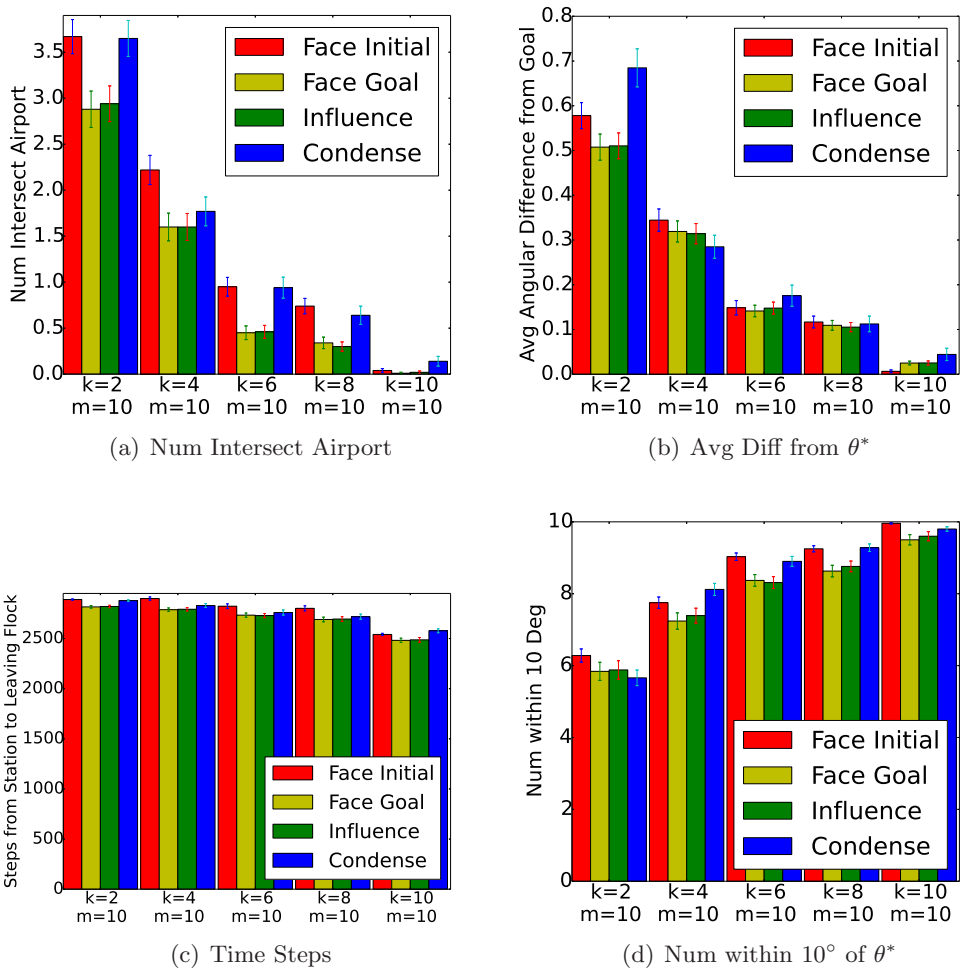
(d) Num within 10° of θ^*

Figure 6.6: Results for different position selection method experiments using *hover* with *face initial* arrival behavior for joining and *hover* for leaving. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

formations perform best in various situations? When should each leaving approach be used?

Let us start by considering the differences between the two *nearest edge* leaving approaches in Figures 6.8 and 6.9. As would be expected, the graphs showing time steps for both leaving approaches look identical — this is because the differences between the *nearest edge* approaches only impact the direction in which the influencing agents leave the flock, which has relatively little impact on the number of time steps required.

We can also compare the difference in the number of flocking agents intersecting the airport in Figures 6.8(a) and 6.9(a). Both leaving approaches result in approximately the same number



(a) Num Intersect Airport

(b) Avg Diff from θ^*

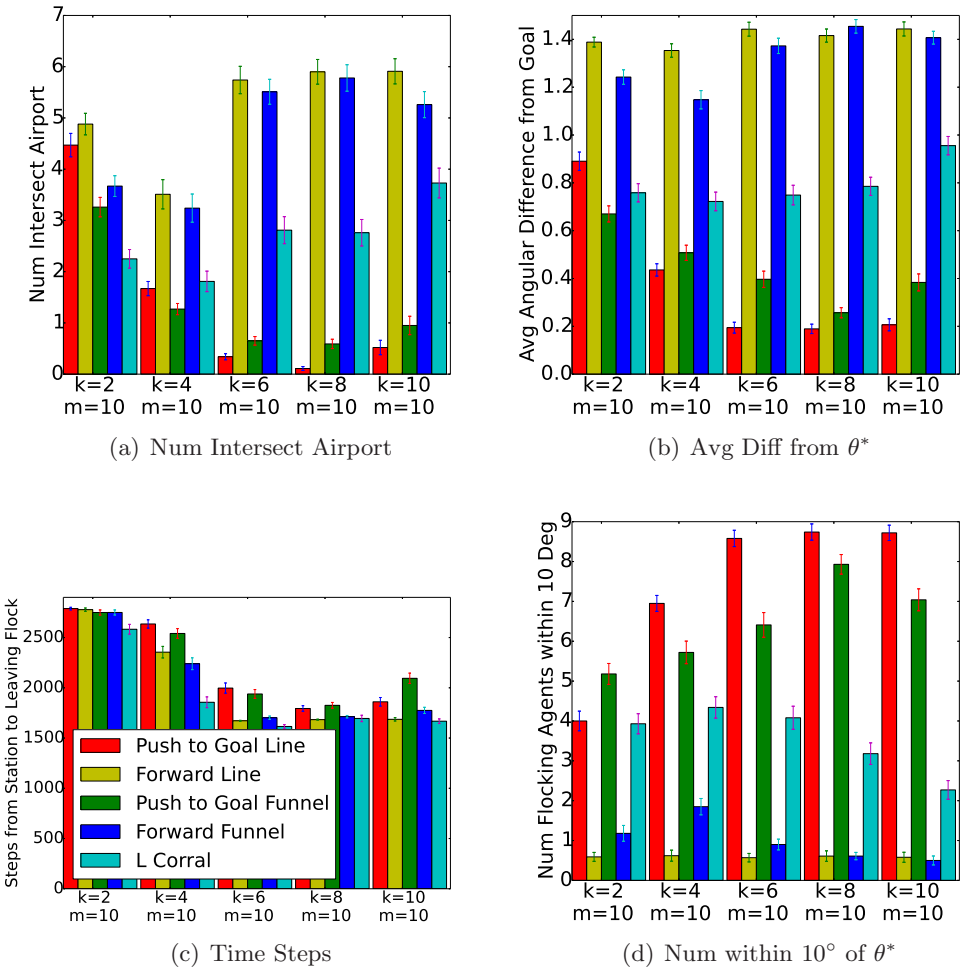
(c) Time Steps

(d) Num within 10° of θ^*

Figure 6.7: Results for arrival behavior experiments using *hover* with *Grid* positions for joining and *hover* for leaving. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

of flocking agents intersecting the airport when $k = 2$. However, the number of flocking agents intersecting the airport increases from $k = 2$ to $k = 6$, $k = 8$, and $k = 10$ in Figure 6.8(a) for the *forward line*, *forward funnel* and *L corral* target formations. This is because the *nearest 3-edge* approach allows influencing agents to leave the flock towards the airport wheres the *nearest 2-edge* approach does not. The influencing agents leaving the flock towards the airport unintentionally influence flocking agents to intersect the airport.

Next we can view Figures 6.8(b) and 6.9(b) to compare the average flocking agent orientation difference from θ^* resulting from using each *nearest edge* leaving approach. *Push to goal line* and



(a) Num Intersect Airport

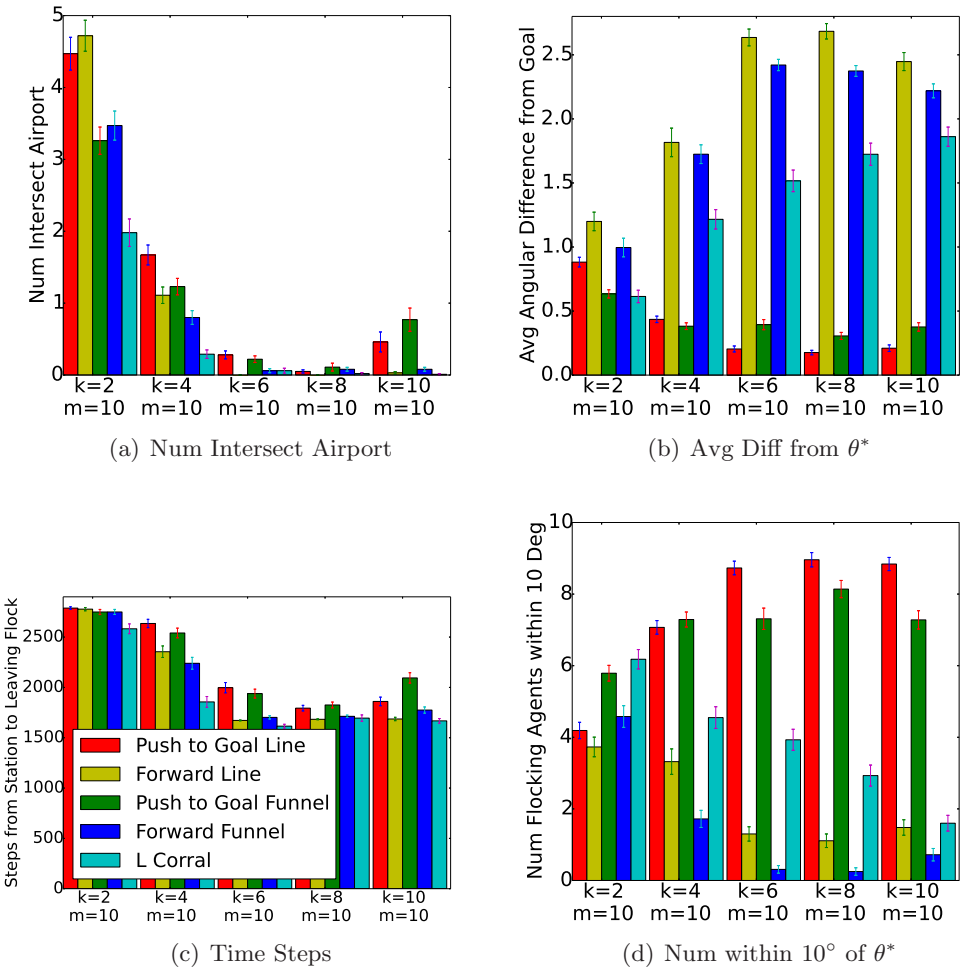
(b) Avg Diff from θ^*

(c) Time Steps

(d) Num within 10° of θ^*

Figure 6.8: Results for different target formations using *intercept* for joining and *nearest 3-edge* for leaving. The legend for all four figures is as depicted in (c). These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

push to goal funnel perform similarly for both *nearest edge* approaches across all k values shown. *Forward line*, *forward funnel*, and *L corral* all perform worse for *nearest 3-edge*. This is because these approaches put influencing agents on the edge of the flock near the airport. Influencing agents that leave towards the airport under the *nearest 3-edge* approach must leave towards the direction from which the flock originally came or away from θ^* under the *nearest 2-edge* approach. Travelling across the flock towards these directions affects more flocking agents than just leaving the flock towards the airport. By affecting more flocking agents, the resulting orientation effect on the flocking agents is dampened.



(a) Num Intersect Airport

(b) Avg Diff from θ^*

(c) Time Steps

(d) Num within 10° of θ^*

Figure 6.9: Results for different target formations using *intercept* for joining and *nearest 2-edge* for leaving. The legend for all four figures is as depicted in (c). These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

Finally, by looking at Figures 6.8(d) and 6.9(d) we can compare the number of flocking agents oriented within 10° of θ^* for each of the *nearest edge* leaving approaches. *Forward line*, *forward funnel*, and *L corral* have more flocking agents within 10° of θ^* when *nearest 2-edge* is utilized — especially for lower values of k. More flocking agents converge within 10° of θ^* for these methods when *nearest 2-edge* is utilized due to the same reasoning expressed in the previous paragraph for Figures 6.8(b) and 6.9(b). Specifically, instead of leaving towards the airport under the *nearest 3-edge* approach, influencing agents on the edge of the flock near the airport must fly across the flock while leaving under the *nearest 2-edge* approach. The influencing agents travelling

across the flock pull more flocking agents away from facing θ^* than when the influencing agents exit towards the airport under the *nearest 3-edge* leaving approach.

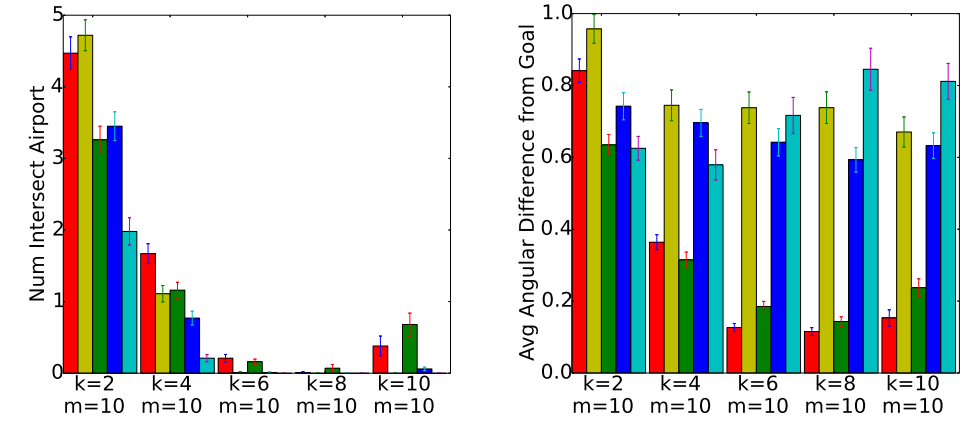
Let us now consider the results in Figure 6.10 for the *influence while leaving* approach. When compared to the *nearest edge* approaches (Figures 6.8 and 6.9), the *influence while leaving* approach performs similarly to the *nearest 2-edge* approach in terms of the number of flocking agents intersecting the airport. The average flocking agent orientation difference from θ^* is less when using the *influence while leaving* approach. To a lesser extent, the number of flocking agents oriented within 10° of θ^* is greater when using the *influence while leaving* approach. Both of these cases in which the *influence while leaving* approach fares better than the *nearest edge* approaches are due to the intentional method of leaving by the *influence while leaving* approach that decreases the impact each leaving influencing agent has on the surrounding flocking agents. One metric in which the more intentional *influence while leaving* approach performs worse is the number of time steps from leaving the charging station to leaving the flock. Since it takes longer for the influencing agents to leave the flock under the *influence while leaving* approach, the time steps are significantly greater in Figure 6.10(c) than in Figures 6.8(c) and 6.9(c).

6.4.3 Discussion

Multiple approaches for joining and leaving a flock in motion were presented in this chapter and analyzed earlier in Section 6.4. In this section, we summarize the main take-away points from this chapter's experiments.

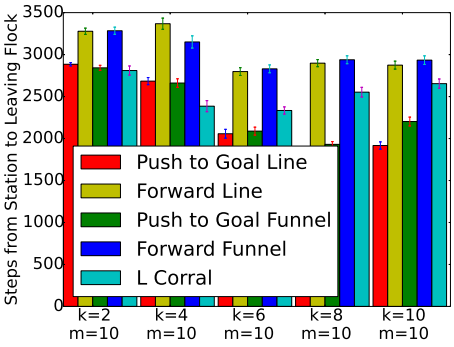
If the influencing agents are not able to hover, it is generally best to use the *push to goal line* target formation (Section 6.1.2) and the *influence while leaving* approach for leaving (Section 6.2.3) if all metrics are equally important. However, if it it critical to minimize the number of flocking agents that intersect with the airport, then it is best to use the *L corral* target formation (Section 6.1.2) and the *influence while leaving* approach for leaving (Section 6.2.3). Similarly, if minimizing the number of time steps between the influencing agents deploying from the charging stations and leaving the flock is critical, then the *push to goal line* target formation (Section 6.1.2) with the *nearest 2-edge* approach for leaving (Section 6.2.2) would be best.

If the influencing agents are able to hover, then *hover* approaches for joining (Section 6.1.1)

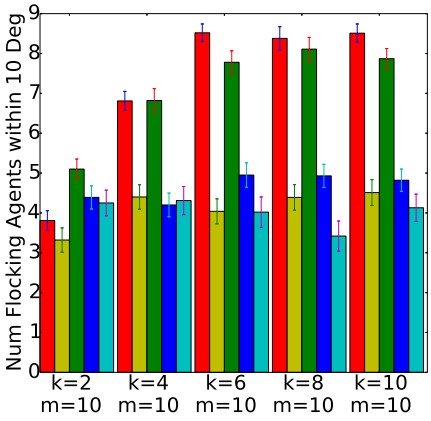


(a) Num Intersect Airport

(b) Avg Diff from θ^*



(c) Time Steps



(d) Num within 10° of θ^*

Figure 6.10: Results for different target formations using *intercept* for joining and *influence while leaving* for leaving. The legend for all four figures is as depicted in (c). These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

and leaving (Section 6.2.1) should be utilized. In general, the *K-Means* position selection method and either the *face goal* or *influence* arrival behavior methods should be used. However, if it is most important to maximize the number of flocking agents oriented within 10° of θ^*, then the *funnel* or *Grid* position selection methods and the *face initial* arrival behavior method should be used.

6.5 Summary

In this chapter, we set out to determine how influencing agents could (1) intercept and join a flock flying towards an airport and (2) leave the flock without influencing the flock to resume its path

towards the airport. This chapter introduced multiple methods for joining a flock in Section 6.1 and multiple methods for leaving a flock in Section 6.2. Since some influencing agents — such as fixed wing robots — might not be able to hover in place, we consider joining and leaving methods that rely on being able to hover in place as well as methods that assume the influencing agents must maintain a non-zero constant velocity. We ran extensive experiments using these joining and leaving methods in a simulated flocking domain. As discussed in Section 6.4.3, our experiments showed that if the influencing agents are able to hover, then *hover* approaches for joining (Section 6.1.1) and leaving (Section 6.2.1) should be utilized. Specifically, the *K-Means* position selection method and either the *face goal* or *influence* arrival behavior methods should be used. If the influencing agents are unable to hover, then the *push to goal line* target formation (Section 6.1.2) and the *influence while leaving* approach for leaving (Section 6.2.3) are likely to be best.

This chapter considered how the influencing agent behaviors described in Chapter 4 could be applied to realistic flocks in which the influencing agents are required to join the flock, influence the flock, and then leave the flock. Chapter 7 will consider how well the algorithms and methods in this dissertation generalize to different neighborhood models and influence models.

7. Evaluation on Different Flocking Models

So far, this dissertation has considered one specific type of flocking behavior. Specifically, most of this dissertation assumes that flocks behave according to a limited version of Reynolds' algorithm for flocking — described in Section 2.1.2 — in which each agent in the flock updates its orientation based on the orientations of its neighbors. Additionally, outside of Chapter 3, this dissertation has also only considered the *visibility radius* neighborhood model described in Section 2.1.1.

However, there are many different types of flocking models that could be utilized. Throughout the work in this dissertation, we wondered how well the methods and algorithms described in this dissertation would generalize to other flocking models — specifically the full Reynolds algorithm for flocking and different neighborhood models. Although in principle the methods described thus far in this dissertation should generalize to the full Reynolds flocking algorithm and to some alternate neighborhood models, whether or not they work in practice is an empirical question. This chapter explores that question.

In this chapter, Section 7.2 describes the alternate neighborhood models that we consider in detail, presents algorithms for determining which agents are within each type of neighborhood, and evaluates how each neighborhood model performs when the *true* neighborhood model — the neighborhood model actually used by the flocking agents — is known or unknown by the influencing agents. Section 7.3 describes Reynolds' algorithm for flocking in detail and then evaluates how using each aspect of Reynolds' algorithm for flocking — both alone and combined with other aspects — performs when the *true* influence model is known or unknown by the influencing agents. As such, the research question addressed in this chapter is: *How well do the algorithms and methods presented so far in the dissertation generalize to alternate influence and neighborhood models when the true model is known or unknown by the influencing agents?*

The work in this chapter assumes that we use the Placement performance metric described in Section 2.2.2 to evaluate performance.

7.1 Experimental Setup

As in Chapters 4, 5, and 6, we use the MASON simulator for our experiments in this chapter. The MASON simulator was introduced in Section 2.3.2, but in this section we review the details of the environment that are important for completely understanding the experimental setup utilized in this chapter. We discuss our experimental setup at the beginning of this chapter so that experiments can be introduced and discussed throughout the chapter. Figure 2.5(c) shows a sample starting configuration for our alternate model experiments. Since we are using the Placement performance metric in this chapter, we use an experimental setup that is very similar to the experimental setup utilized for our placement experiments in Chapter 5.

The relevant experimental variables for our alternate model experiments are given in Table 7.1.

Variable	**Value**
toroidal domain	no
domain height	300
domain width	300
units moved by each agent per time step (v_i)	0.2

Table 7.1: Experimental variables for our alternate model experiments.

None of the experimental variables in Table 7.1 are set to the default settings for the MASON simulator. We removed the toroidal nature of the domain in order to make the domain more realistic. Hence, if an agent moves off of an edge of our domain, it will not reappear. This is particularly important for *lost* agents remaining *lost*. We also increased the domain height and width, and decreased the units each agent moves per time step, in order to give agents a chance to converge with the flock before leaving the visible area. However, we have no reason to believe the exact experimental settings we chose for our experiments are of particular importance.

All of the experiments reported in this chapter use $m = 10$ flocking agents and $k = 2$ to $k = 10$ influencing agents. During our initial research, we ran a small number of larger scale experiments with $m = 50$ flocking agents and $k = 10$ influencing agents. In these experiments, we found that most results were similar for our $m = 10$ experiments — as such, we only present results for experiments with $m = 10$ flocking agents.

Figure 2.5(c) shows an example starting configuration for the experiments in this chapter. For the experiments in this chapter, the flocking agents are initially randomly placed within FA_{preset}, which is a small square at the top left of the environment. All of the flocking agents are initially assigned random headings that are within 90 degrees of θ^*.

Experimental results and discussion will be presented throughout the following sections. In all of our experiments, we run 100 trials for each experimental setting and we use the same set of 100 random seeds for each set of experiments. The random seeds are used to determine the exact placement and orientation of all of the flocking agents at the start of a simulation experiment. The error bars in all of our graphs depict the standard error of the mean. Based upon the Placement performance metric described in Section 2.2.2, throughout this chapter we only present results for the average number of flocking agents *lost*.

7.2 Alternate Neighborhood Models

As discussed in Section 2.1.1, most flocking models state that flocking agents are only influenced by other agents that are located within their *neighborhood*. Throughout much of this dissertation, we utilized the *visibility radius* neighborhood model described in Section 2.1.1. This *visibility radius* neighborhood model served as a simple approximation of the neighborhood model generally believed to be utilized by real-life birds. However, some biologists claim that most birds are actually influenced by the six or seven nearest neighbors [6, 13]. Additionally, there are other neighborhood models that are technologically feasible to implement for robot birds. In this section, we consider multiple alternate neighborhood models. Each alternate neighborhood model we consider is either biologically plausible for real birds or technologically feasible to implement for robot birds. In Section 7.2.4 we present results from experiments using all of these neighborhood models.

Although all of the alternate neighborhood models we consider in this section were introduced in Section 2.1.1, in this section we present algorithms for determining which agents lie within an agent's neighborhood under these neighborhood models. Special notation is utilized in these algorithms — this notation is depicted in Table 7.2. When referencing Table 7.2, keep in mind the following definitions: *list* is a collection of elements, x is a number, *l1* and *l2* are locations, and

element is an object in the environment.

Notation	Meaning
element.loc	Returns the x,y location of *element*
calcDist(l1, l2)	Returns the distance between locations *l1* and *l2*
list.append(element)	Appends *element* to the end of *list*
list.get(x)	Returns the *x*th element in 0-indexed *list*
list.remove(element)	Removes all instances of *element* from *list*
list.remove(x)	Removes the *x*th element in a 0-indexed *list*
list.size()	Returns the number of elements in *list*
MAXVALUE	Returns the max value representable within the programming environment
random.nextDouble()	Returns a random decimal number greater than or equal to 0.0 and less than 1.0

Table 7.2: Notation utilized in Algorithms 11, 12, and 13.

7.2.1 Visibility Sector

In Chapter 3, a *visibility sector* was used to define each agent's neighborhood. However, a *visibility radius* neighborhood model was used throughout the other chapters in this dissertation. Hence, in this section we re-introduce the concept of a *visibility sector* neighborhood model and show in Algorithm 11 how we calculate which agents lie within a *visibility sector* of less than π radians. Most robots have a limited number of cameras, each with a limited field of view. This field of view usually does not cover 360° (2π radians), so a *visibility sector* is an especially realistic neighborhood model. This is especially true for the types of robot birds that could be used to influence flocks, as these robots would likely only have one camera with a field of view that is less than 180°.

Algorithm 11 considers all of the agents within a particular radius r of agent a_i. The algorithm takes in the visibility sector angle *visAngle* and the current orientation $\theta_i(t)$ of agent a_i at time t. Both inputs are in radians. The algorithm calculates whether each agent within r also falls within *visAngle* and returns a list *agents* containing these agents. Specifically, the algorithm computes the points of an isosceles triangle that covers the area of overlap between radius r and the visibility sector angle *visAngle* — see Figure 7.1 for a visual depiction of the coverage area. This coverage is guaranteed because the triangle has a perpendicular bisector of length r and hence actually also covers some area outside of r. The three points of the isosceles triangle are referred to as *p1*, *p2*, and *p3* in Algorithm 11.

Now we walk through Algorithm 11. The side length *sidelength* of the isosceles triangle is

Algorithm 11 agents = getVisibilitySectorNeighbors(visAngle, $\theta_i(t)$)

radNeigh ← getRadiusNeighbors(), counter ← 0
while counter < radNeigh.size() **do**
  curAgent ← radNeigh.get(counter), remove ← true
  p ← currentAgent.loc, p1 ← a_i.loc, p2 ← (0.0,0.0), p3 ← (0.0,0.0)
  sidelength ← $\sqrt{r^2 + (\frac{r * \frac{\text{visAngle}}{2}}{\sin(\frac{\pi}{2} - \frac{\text{visAngle}}{2})})^2}$
  lowOrient ← $(\theta_i(t) - \frac{\text{visAngle}}{2})\%2\pi$, lowSlope ← tan(lowOrient)
  lowX1 ← sidelength $* \frac{1}{\sqrt{1+(\text{lowSlope}^2)}} + a_i$.loc.x, lowX2 ← a_i.loc.x − sidelength $* \frac{1}{\sqrt{1+(\text{lowSlope}^2)}}$
  lowY1 ← sidelength $* \frac{\text{lowSlope}}{\sqrt{1+(\text{lowSlope}^2)}} + a_i$.loc.y, lowY2 ← a_i.loc.y − sidelength $* \frac{\text{lowSlope}}{\sqrt{1+(\text{lowSlope}^2)}}$
  greaterX ← 0.0, lesserX ← 0.0, greaterY ← 0.0, lesserY ← 0.0
  if lowX1 < lowX2 **then**
    greaterX ← lowX2, lesserX ← lowX1
  else
    greaterX ← lowX1, lesserX ← lowX2
  if lowY1 < lowY2 **then**
    greaterY ← lowY2, lesserY ← lowY1
  else
    greaterY ← lowY1, lesserY ← lowY2
  if lowOrient ≤ $\frac{\pi}{2}$ **then**
    p2.x ← greaterX, p2.y ← greaterY
  else if lowOrient ≤ π **then**
    p2.x ← lesserX, p2.y ← greaterY
  else if lowOrient ≤ $\frac{3\pi}{2}$ **then**
    p2.x ← lesserX, p2.y ← lesserY
  else
    p2.x ← greaterX, p2.y ← lesserY
  highOrient ← $(\theta_i(t) + \frac{\text{visAngle}}{2})\%2\pi$, highSlope ← tan(*highOrient*)
  highX1 ← sidelength $* \frac{1}{\sqrt{1+(\text{highSlope}^2)}} + a_i$.loc.x, highX2 ← a_i.loc.x − sidelength $* \frac{1}{\sqrt{1+(\text{highSlope}^2)}}$
  highY1 ← sidelength $* \frac{\text{highSlope}}{\sqrt{1+(\text{lowSlope}^2)}} + a_i$.loc.y, highY2 ← a_i.loc.y − sidelength $* \frac{\text{lowSlope}}{\sqrt{1+(\text{highSlope}^2)}}$
  if highX1 < highX2 **then**
    greaterX ← highX2, lesserX ← highX1
  else
    greaterX ← highX1, lesserX ← highX2
  if highY1 < highY2 **then**
    greaterY ← highY2, lesserY ← highY1
  else
    greaterY ← highY1, lesserY ← highY2
  if highOrient ≤ $\frac{\pi}{2}$ **then**
    p3.x ← greaterX, p3.y ← greaterY
  else if highOrient ≤ π **then**
    p3.x ← lesserX, p3.y ← greaterY
  else if highOrient ≤ $\frac{3\pi}{2}$ **then**
    p3.x ← lesserX, p3.y ← lesserY
  else
    p3.x ← greaterX, p3.y ← lesserY
  $\alpha \leftarrow \frac{(\text{p2.y}-\text{p3.y})*(\text{p.x}-\text{p3.x})+(\text{p3.x}-\text{p2.x})*(\text{p.y}-\text{p3.y})}{(\text{p2.y}-\text{p3.y})*(\text{p1.x}-\text{p3.x})+(\text{p3.x}-\text{p2.x})*(\text{p1.y}-\text{p3.y})}$
  $\beta \leftarrow \frac{(\text{p3.y}-\text{p1.y})*(\text{p.x}-\text{p3.x})+(\text{p1.x}-\text{p3.x})*(\text{p.y}-\text{p3.y})}{(\text{p2.y}-\text{p3.y})*(\text{p1.x}-\text{p3.x})+(\text{p3.x}-\text{p2.x})*(\text{p1.y}-\text{p3.y})}$
  $\gamma \leftarrow 1.0 - \alpha - \beta$
  if $\alpha \geq 0$ && $\alpha \leq 1.0$ && $\beta \geq 0$ && $\beta \leq 1.0$ && $\gamma \geq 0$ && $\gamma \leq 1.0$ **then**
    remove ← false
  else
    remove ← true
  if remove || (p.x == p1.x && p.y == p1.y) **then**
    radNeigh.remove(curAgent)
  else
    i++
end while
return radNeigh

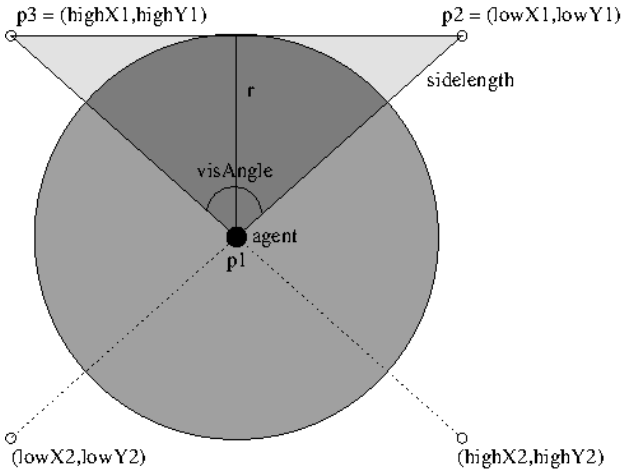


Figure 7.1: A figure depicting how Algorithm 11 selects agents in a *visibility sector* (darkest shade) from the agents in a *visibility radius* (moderate shade) using an isosceles triangle (lightest shade). Note that (highX1,highY1) and (highX2,highY2) (as well as (lowX1,lowY1) and (lowX2,lowY2)) may be swapped depending on the dynamics of the environment.

calculated on line 5. Lines 6–25 find point *p2*. Specifically, line 6 calculates the heading *lowOrient* and slope *lowSlope* of one of the sides of the triangle. These calculations on line 6 then allow the two possible points for *p2* to be calculated on lines 7 and 8. As shown in Figure 7.1, there are two points because there are two points that lie along slope *lowSlope* that are *sidelength* away from agent a_i. Lines 10–25 then determine which of these two x values and two y values make up *p2*. In particular, lines 10–17 order the x and y values based upon their magnitude and lines 18–25 set point *p2* based upon the quadrant that *lowOrient* falls within. In a similar manner, lines 26–44 find point *p3*. Note that point *p3* could also be found by reflection over the radial line — but for simplicity we used the same method that we used to find point *p2*. Lines 45–51 use points *p1*, *p2*, and *p3* to determine whether point p lies within the triangle made up of points *p1*, *p2*, and *p3*. In particular, on lines 45–47 we calculate the barycentric coordinates (α, β, and γ) of point p with respect to the triangle formed from points *p1*, *p2*, and *p3*. If each of the barycentric coordinates is greater than or equal to zero and less than or equal to one, then point p is known to lie within the sector formed by points *p1*, *p2*, and *p3*.

Note that although the *visAngle* input to Algorithm 11 should be less than π radians, we can handle larger *visAngle* inputs in a pre-processing step. Specifically, we can input $2\pi - visAngle$ and remove the returned *agents* output from *getRadiusNeighbors()*. In the case where *visAngle* is

exactly π, we make the simple approximation that *visAngle* is $\pi - 0.01$ in order to avoid handling this special case in a more complicated manner.

7.2.2 N-Nearest Neighbors

Although the *visibility sector* model is the most technologically feasible neighborhood model for robot birds, biologists claim that birds are actually influenced by their six or seven nearest neighbors [6, 13]. With this in mind, in this section we introduce a *N-nearest neighbors* neighborhood model in Algorithm 12.[1] This algorithm identifies the N neighbors that are currently closest to agent a_i.

Algorithm 12 agents = getNNearestNeighbors(N, allAgents, includeSelf)

```
 1: agents ← {}
 2: for numAdded=0; numAdded < N; ++numAdded do
 3:     minDist ← MAXVALUE
 4:     minIndex ← −1
 5:     for currentNeighbor=0; currentNeighbor < allAgents.size(); ++currentNeighbor do
 6:         distToCurrSq ← calcDist(a_i.loc, allAgents.get(currentNeighbor).loc)
 7:         if distToCurrSq < minDist then
 8:             if distToCurrSq > 0 || includeSelf then
 9:                 minDist ← distToCurrSq
10:                 minIndex ← currentNeighbor
11:     agents.append(allAgents.get(minIndex))
12:     allAgents.remove(minIndex)
13: return agents
```

Algorithm 12 takes in the number of neighbors to return (N), all of the agents to consider as potential neighbors (*allAgents*), and a boolean that indicates whether agent a_i should include itself as a neighbor (*includeSelf*). Lines 2–12 consider each potential neighbor N times. On each iteration, line 11 adds the closest neighbor that is not already in *agents* to *agents*. After N neighbors have been added to *agents*, *agents* is returned on line 13.

7.2.3 Weighted Influence

The *weighted influence* neighborhood model considers the idea that closer neighbors should have more influence than farther neighbors. Under the *weighted influence* neighborhood model, the overall influence exerted is the same as in the *visibility radius* model but closer neighbors have more influence while farther neighbors have less influence.

[1] This algorithm was implemented by Basil Hariri, an undergraduate student whom I mentored for two semesters.

The neighbors obtained for an agent a_i that uses the *weighted influence* neighborhood model are the same as the neighbors obtained by Algorithm 11 for an agent that uses the *visibility radius* neighborhood model. However, all of the neighbors are weighted equally under the *visibility radius* neighborhood model, while each neighbor is weighted differently under the *weighted influence* model. The methodology of assigning weights to neighbors is presented in Algorithm 13.

Algorithm 13 agents = getDecayingInfluenceNeighbors(neighbors)

```
1:  weights ← ()
2:  totalInfluence ← neighbors.size()
3:  totalDistance ← 0
4:  for counter=0; counter < neighbors.size(); counter++ do
5:      neighborLoc ← neighbors.get(counter).loc
6:      if |a_i.loc.x − neighborLoc.x| > 0.0001 && |a_i.loc.y − neighborLoc.y| > 0.0001 then
7:          totalDistance ← totalDistance + neighborhood − sqrt((a_i.loc.x-neighborLoc.x)^2 + (a_i.loc.y-neighborLoc.y)^2)
8:      else
9:          totalInfluence ← totalInfluence - 1
10: for counter=0; counter < neighbors.size(); counter++ do
11:     neighborLoc ← neighbors.get(counter).loc
12:     if |a_i.loc.x − neighborLoc.x| > 0.0001 && |a_i.loc.y − neighborLoc.y| > 0.0001 then
13:         weights.add(totalInfluence * (neighborhood − sqrt((a_i.loc.x − neighborLoc.x)^2 + (a_i.loc.y − neighborLoc.y)^2)) / totalDistance)
14:     else
15:         weights.append(1.0)
16: return weights
```

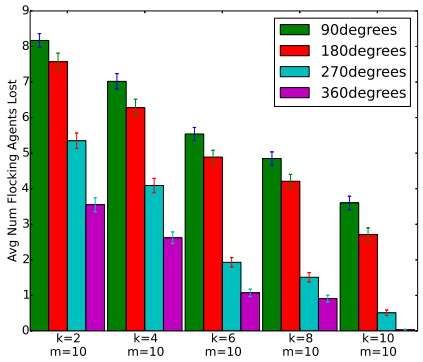
Algorithm 13 takes in an ordered list of neighbors *neighbors* and the neighborhood size *neighborhood*. Lines 4–9 add up the total distance of all of the neighbors from agent a_i (*totalDistance*) and calculate the total influence to be distributed. Then lines 10–15 set the weight for each neighbor based on its distance from a_i, the neighborhood size *neighborhood*, and the total distance *totalDistance*. The algorithm returns an ordered list of weights *weights*. Since both *neighbors* and *weights* are ordered, weights.get(x) returns the weight associated with neighbors.get(x), where x is an integer within (0,...,neighbors.size()-1).

7.2.4 Experimental Results

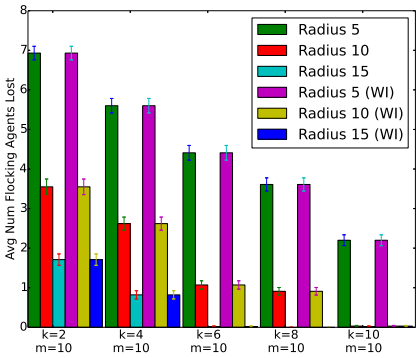
In this section, we present experiments that compare the performance of the four neighborhood models: *visibility radius*, *visibility sector*, *N-nearest neighbors*, and *weighted influence*.

Figure 7.2 shows results for different neighborhood models when influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2, while Figure 7.3 shows

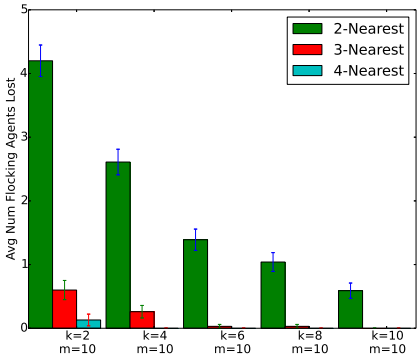
results for different neighborhood models when influencing agents are originally positioned using the *Border* placement method from Section 5.2.3.



(a) Visibility Sector



(b) Visibility Radius and Weighted Influence

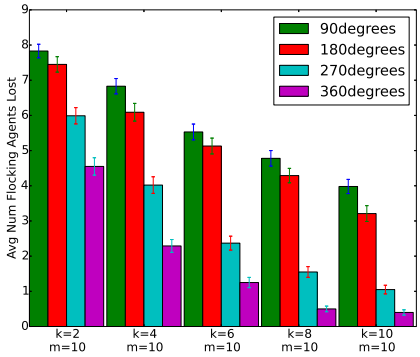


(c) N-Nearest

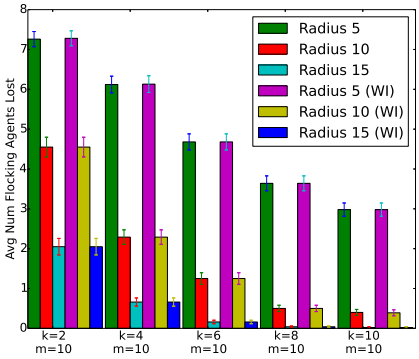
Figure 7.2: Results for different neighborhood models when influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean. WI stands for *weighted influence*.

Figures 7.2 and 7.3 show similar trends. As we would expect, in (a) we see larger visibility sectors losing fewer flocking agents on average than smaller visibility sectors, in (b) we see larger visibility radii losing fewer flocking agents on average than smaller visibility radii, and in (c) we see that using a greater N for *N-nearest neighbors* loses fewer flocking agents on average.

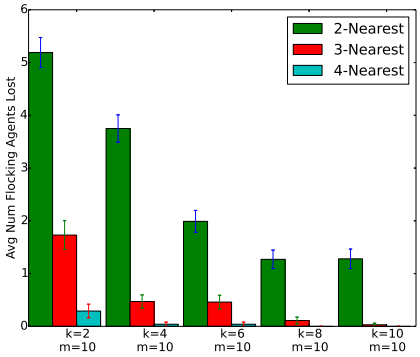
There are a few interesting trends to note in Figures 7.2 and 7.3. First, in (a) the difference between the results of a visibility sector of 90 degrees and 180 degrees is much smaller than the difference between visibility sectors of 180 degrees and 270 degrees. This is likely due to how the



(a) Visibility Sector



(b) Visibility Radius and Weighted Influence



(c) N-Nearest

Figure 7.3: Results for different neighborhood models when influencing agents are originally positioned using the *Border* placement method from Section 5.2.3. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean. WI stands for *weighted influence*.

influencing agents are spread across the environment under the *Grid* and *Border* placement methods. Second, in (c), using the 3-Nearest neighbors instead of the 2-Nearest neighbors significantly decreases the average number of flocking agents lost. This is due to the substantially quicker spread of influence when using 3-Nearest neighbors instead of 2-Nearest neighbors.

The most unexpected result from Figures 7.2 and 7.3 is certainly in (b). We had expected to see a noticeable difference between the *visibility radius* neighborhood and the *weighted influence* neighborhood. Although both the *visibility radius* neighborhood and the *weighted influence* neighborhood contain exactly the same neighbors, we expected that giving closer neighbors more influence would result in fewer flocking agents becoming lost. However, there is no noticeable dif-

ference between the *visibility radius* and *weighted influence* results in Figures 7.2(b) and 7.3(b) — although the exact average number of flocking agents lost is slightly different. After considering the results more carefully and observing some trials, the *visibility radius* neighborhood and the *weighted influence* neighborhood end up performing similarly because (1) both neighborhood models have the exact same neighbors and (2) in our domain there is no inherent reason to "trust" closer neighbors more than farther neighbors.

Performance with Incorrect Neighborhood Models

Although we found in the previous section that some neighborhood models result in fewer flocking agents becoming *lost*, we cannot control which neighborhood model the flocking agents utilize. In fact, the influencing agents may not know which neighborhood model the flocking agents are utilizing. If we consider our motivating example of adding robot birds to a flock of real birds, it is likely that the robot birds will not know the exact neighborhood model utilized by the real birds. With this in mind, in this section we consider the performance of various models when the influencing agents do not know the *true* neighborhood model currently being utilized by the flocking agents.

For our experiments, we gathered results in which the influencing agents are originally positioned using both the *Grid* placement method from Section 5.2.2 and the *Border* placement method from Section 5.2.3 — but since both sets of results are similar, we only present results for experiments using the *Grid* placement method. Additionally, since the previous section found that the *visibility radius* and *weighted influence* neighborhood models performed almost identically, we do not present results for the *weighted influence* neighborhood model in this section.

Figures 7.4, 7.5, and 7.6 show results when the flocking agents are utilizing a variety of neighborhood models. In each figure, we show the *true* neighborhood model — i.e., the neighborhood model being utilized by the flocking agents — as well as (1) variants of the *true* neighborhood model and (2) the best performing variant of the other neighborhood models. The remainder of this section considers each of these figures separately.

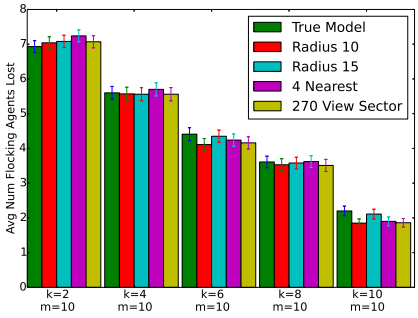
Figure 7.4 considers different *visibility radius* neighborhood models as the *true* neighborhood model. In Figure 7.4(a) — when the flocking agents are behaving according to the *visibility radius*

neighborhood model where $r = 5$ — there is little harm in the influencing agents behaving according to any of the other neighborhood models tested. However, Figure 7.4(b) shows that when the flocking agents behave according to the *visibility radius* neighborhood model where $r = 10$, there are some cases where more flocking agents will become lost on average if the influencing agents do not assume the correct neighborhood model. Finally, Figure 7.4(c) shows that when the flocking agents behave according to the *visibility radius* neighborhood model where $r = 15$, it is important that the influencing agents assume the *true* neighborhood model when $k > 6$.

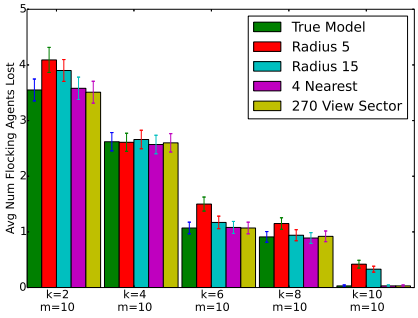
As seen in Figures 7.4(b) and 7.4(c), poor performance occurs — especially for larger values of k — when the influencing agents assume a smaller r than is actually utilized by the flocking agents. This is because in these cases some of the influencing agents are within the neighborhood of the flocking agents without realizing it and hence influence the flocking agents unintentionally. It is harmful for the influencing agents to not realize they are influencing the flocking agents because then the influencing agents are not acting to influence nearby agents (and may instead be attempting to leave the flock or reposition). It is much less harmful for the influencing agents to believe they are influencing flocking agents when they are not, as then the influencing agents simply have no influence.

Figure 7.5 considers different *visibility sector* neighborhood models as the *true* neighborhood model. In Figure 7.5(a) — when the flocking agents are behaving according to the *visibility sector* neighborhood model where $\alpha = 90°$ — significantly fewer flocking agents are lost on average when the influencing agents assume the *true* neighborhood model. For most k, if the influencing agents incorrectly believe the flocking agents are utilizing the *visibility sector* neighborhood model where $\alpha = 180°$, they perform significantly worse than if they had used the *true* neighborhood model but significantly better than if they had assumed the flocking agents were utilizing any other neighborhood model.

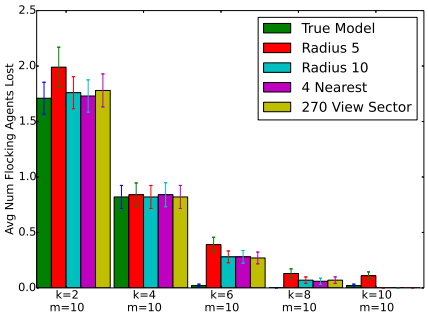
In Figures 7.5(b) and 7.5(c) — when the flocking agents are behaving according to the *visibility sector* neighborhood model where $\alpha = 180°$ and $\alpha = 270°$ — most of the neighborhood models perform approximately the same. However, in both figures performance is significantly worse if the influencing agents incorrectly assume the flocking agents are utilizing a $\alpha = 90°$ *visibility sector*. This is because in these cases — much like in Figures 7.4(b) and 7.4(c) — the



(a) True model: $r = 5$



(b) True model: $r = 10$



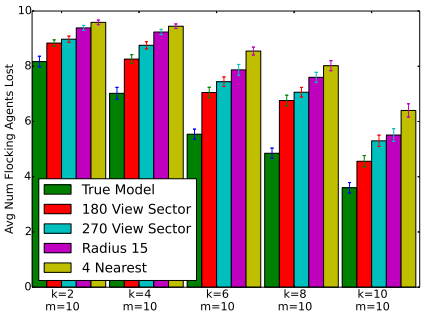
(c) True model: $r = 15$

Figure 7.4: Results for using different *visibility radius* neighborhood models as the *true* neighborhood model when influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

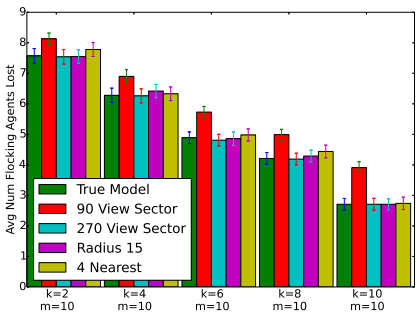
influencing agents will unintentionally influence the flocking agents.

Figure 7.6 considers different *N-nearest neighbors* neighborhood models as the *true* neighborhood model. In Figures 7.6(a), 7.6(b), and 7.6(c) we see that using an incorrect neighborhood model never performs significantly worse than using the correct model. This is because when the flocking agents are behaving according to the *N-nearest neighbors* neighborhood models, almost any other model assumed by the influencing agents will result in the influencing agents believing they are neighbors of a flocking agent when the flocking agent does not see them as neighbors. More importantly, the influencing agents will very rarely incorrectly believe they are not influencing the flocking agents — meaning the influencing agents will rarely unintentionally influence the flocking agents due to an incorrect model.

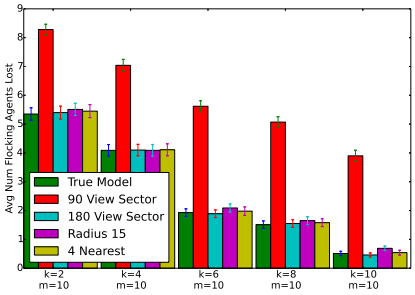
In Figure 7.6(c) the average number of flocking agents lost was zero for all models when



(a) True model: $\alpha = 90°$



(b) True model: $\alpha = 180°$

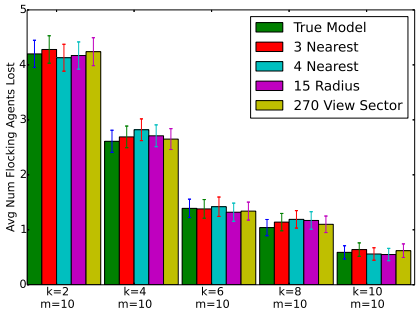


(c) True model: $\alpha = 270°$

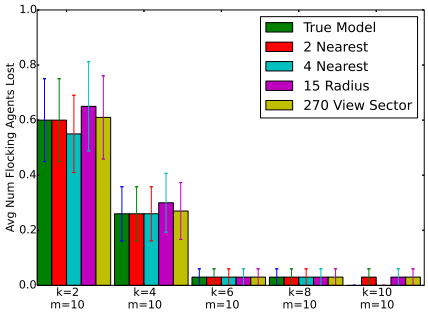
Figure 7.5: Results for using different *visibility sector* neighborhood models as the *true* neighborhood model when influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

$k > 2$. This is because when $N = 4$ each flocking agent considered the four nearest agents to be neighbors — and this caused influence from the influencing agents to spread throughout the flock quickly even if the flock became dispersed.

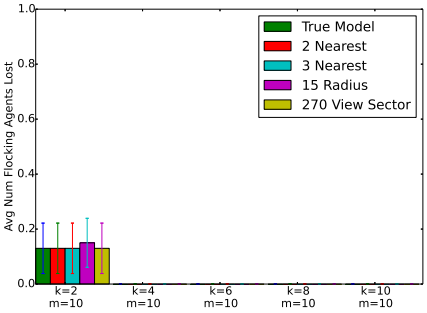
There are a few take-away points for this section. First, we found that the algorithms and methods presented in previous chapters of this dissertation generalize to the neighborhood models discussed in Section 2.1.1. Second, we found that — especially with the *N-nearest neighbors* neighborhood models — it is not necessarily important that the influencing agents know the exact neighborhood model of the flocking agents. In the next section, we consider whether the algorithms and methods presented earlier in the dissertation also generalize to alternate influence models.



(a) True model: $N = 2$



(b) True model: $N = 3$



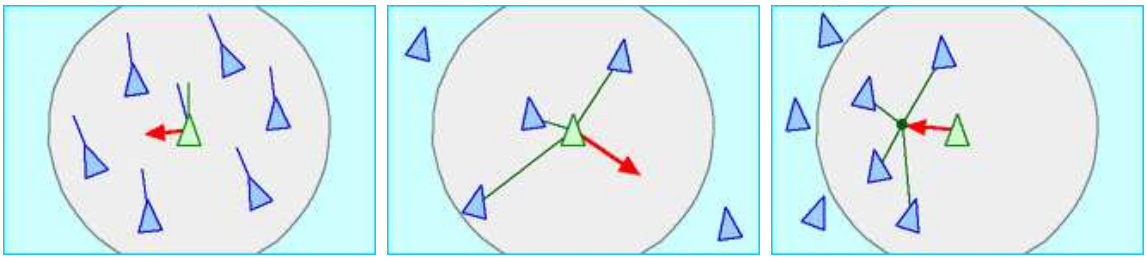
(c) True model: $N = 4$

Figure 7.6: Results for using different *N-nearest neighbors* neighborhood models as the *true* neighborhood model when influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

7.3 Alternate Influence Models

The complete Reynolds' Boid algorithm for flocking is described in Section 2.1.2. As noted in Section 2.1.2, the work presented so far in this dissertation only utilized the *alignment* aspect of Reynolds' algorithm for flocking. Remember from Section 2.1.2 that the *alignment* aspect steers each agent towards the average heading of its neighbors. Additionally, the *separation* aspect steers each agent away from its neighbors to avoid collisions and the *cohesion* aspect steers each agent towards the average position of its neighbors. See Figure 7.7 for pictorial descriptions of each aspect of Reynolds' Boid algorithm for flocking.

Our decision to utilize only the *alignment* aspect throughout most of this dissertation was mainly for the purpose of simplicity. The *alignment* aspect alone resulted in stable flocking, so we used the simplest global orientation update possible for the flocking agents (see Equation 1)



(a) Alignment (b) Separation (c) Cohesion

Figure 7.7: Pictorial descriptions of each aspect of Reynolds' Boid algorithm for flocking.[2]

by utilizing only the *alignment* aspect. Although we believed that the algorithms and methods in this dissertation would apply to the complete Reynolds' algorithm for flocking, in this section we determine whether our intuition was correct.

7.3.1 Experimental Results

In this section, we present experiments that compare the performance of seven influence models. Each of the influence models corresponds to one of the seven possible combinations of the three aspects of Reynolds' algorithm for flocking: *Alignment*, *Separation*, *Cohesion*, *Alignment* + *Separation*, *Alignment* + *Cohesion*, *Separation* + *Cohesion*, and *Alignment* + *Separation* + *Cohesion*.

Figure 7.8 compares the performance of all seven influence models. In Figure 7.8, the flocking agents are using the influence model noted in the legend and the influencing agents are also aware that the flocking agents are using this influence model. In other words, in Figure 7.8 the influencing agents know the *true* influence model.

The most important and noticeable trend in Figure 7.8 is that for all k, *Alignment* loses the fewest flocking agents on average, followed by *Alignment* + *Cohesion* and *Alignment* + *Separation* + *Cohesion*. Notice that all of the combinations that are not at least partially comprised of the *Alignment* aspect do rather poorly. These results show that the *Alignment* aspect is critical for minimizing the number of lost flocking agents, since the *Alignment* aspect generally keeps the flock together and moving with the same orientation. Note that the *Alignment* + *Separation* combination performs poorly, likely due to the *Separation* component breaking the flock apart.

Since the *Alignment* aspect performs well given our primary *Placement* performance metric

[2]By Craig Reynolds, `http://www.red3d.com/cwr/boids/`

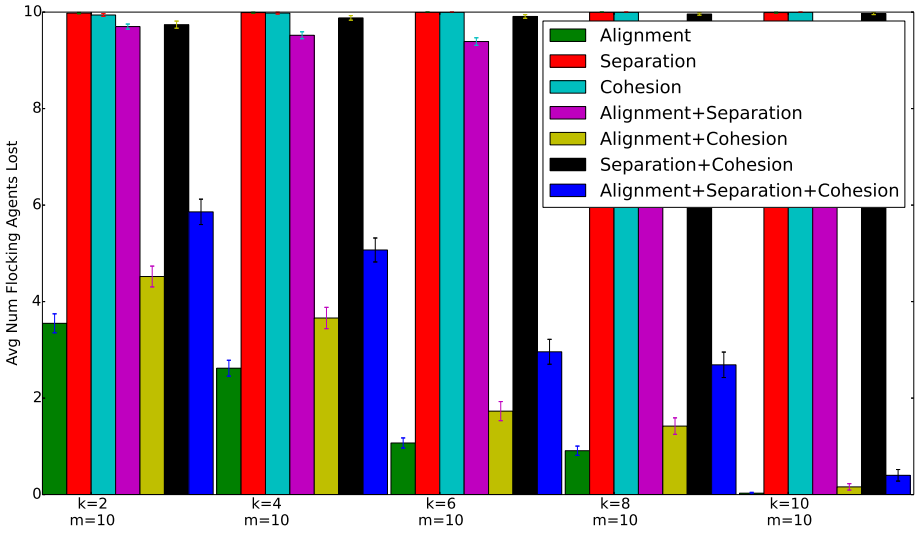


Figure 7.8: Results using all seven combinations of the three aspects of Reynolds' algorithm for flocking as the *true* influence model. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

(Section 2.2.2) of minimizing the number of lost flocking agents, we can conclude that using only the *Alignment* aspect was a reasonable choice for the majority of this dissertation. With this conclusion made, in the next section we consider how important it is for the influencing agents to know the *true* influence model being utilized by the flocking agents.

Performance with Incorrect Influence Models

Section 7.2.4 considered how performance was affected when the influencing agents did not know the true *neighborhood model* of the flocking agents. In this section, we present results for a similar experiment. In particular, we consider how performance is affected when the influencing agents do not know the true *influence model* of the flocking agents.

For each figure in this section, a particular combination of the aspects of Reynolds' algorithm for flocking will be the *true* influence model — or in other words, the influence model that is utilized by the flocking agents — and we will consider how performance differs based on what influence model the influencing agents believe the flocking agents are utilizing.

Figure 7.9 shows results when the flocking agents are using the *Alignment* influence model.

Each of the different colors in the legend corresponds to the influence model that the influencing agents believe is being used by the flocking agents. As might be expected, the best performance is obtained when the influencing agents believe the flocking agents are utilizing the *Alignment* influence model. In other words, the influencing agents know the *true* influence model. The other combinations that include the *Alignment* aspect perform better than combinations without the *Alignment* aspect, which is expected since the *Alignment* aspect keeps the flocking agents' orientations and locations together.

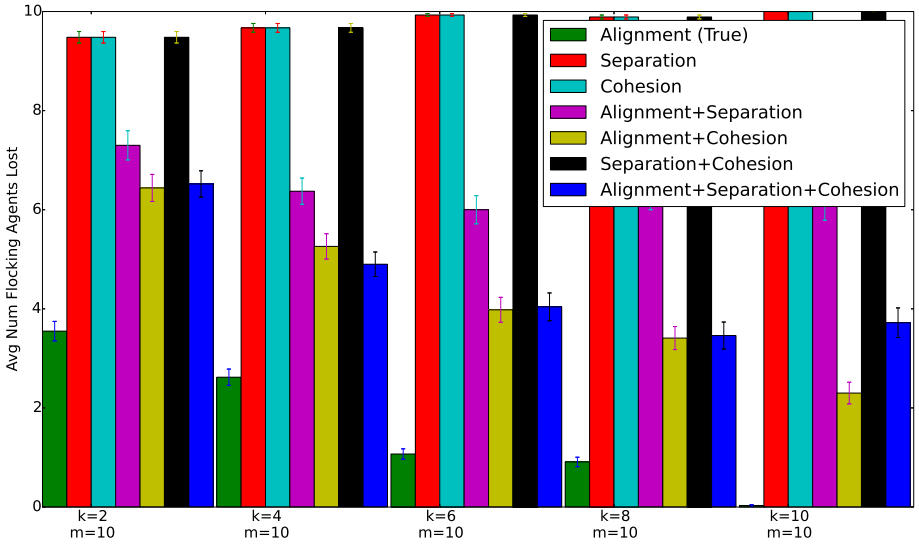


Figure 7.9: Results when *Alignment* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

Figure 7.10 shows results when the flocking agents are using the *Separation* influence model. *Separation* causes each flocking agent to avoid collisions with nearby neighbors — this type of behavior generally causes the flock to splinter which inherently results in many flocking agents becoming *lost*. As can be seen in Figure 7.10, performance is best when $k \geq 6$ and the influencing agents incorrectly believe the flocking agents are using the *Alignment* influence model. This is because the behavior of a significant number (greater than or equal to six, in this case) of influencing agents that behave as if the flocking agents are using the *Alignment* influence model can influence

a couple of flocking agents on average to stay together and orient towards the goal.

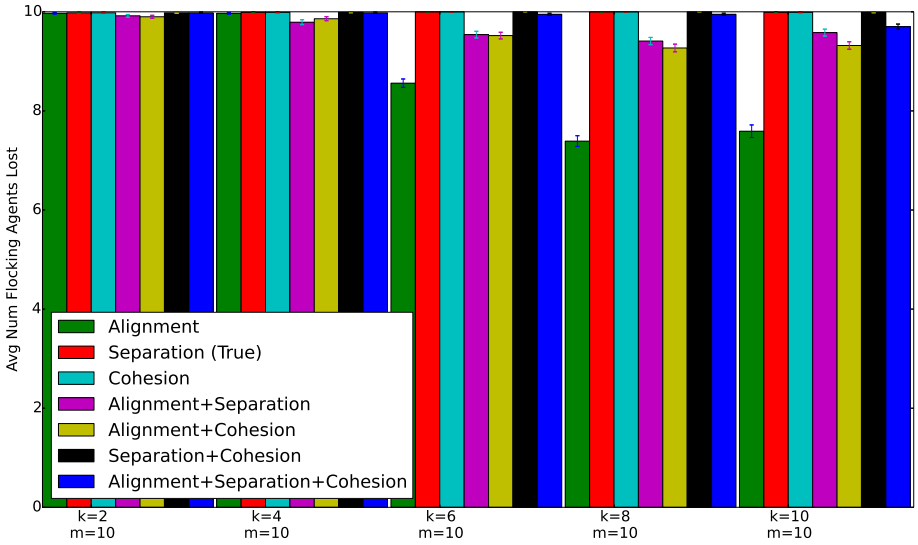


Figure 7.10: Results when *Separation* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

Figure 7.11 shows results when the flocking agents are using the *Cohesion* influence model. *Cohesion* causes each flocking agent to steer towards the center location of its neighbors. This type of behavior tends to keep the flock together. In Figure 7.11 we notice that the best performance is obtained when the influencing agents believe the flocking agents are using the *Alignment* or *Alignment* + *Cohesion* influence models. This is because in both cases, the influencing agents behave to keep the flock together and oriented towards the goal — hence, losing fewer flocking agents.

Figure 7.12 shows results when the flocking agents are using the *Alignment* + *Separation* influence model. *Alignment* keeps the flocking agents orienting together while *Separation* pushes the flock apart. However, *Alignment* + *Separation* together sends the flock slowly moving apart but towards similar orientations. In Figure 7.12, we see that performance is generally bad for all influencing models — but is slightly better for $k \geq 6$. Similarly to Figure 7.10 in which the flocking agents behaved according to just the *Separation* influence model, in this case it takes multiple

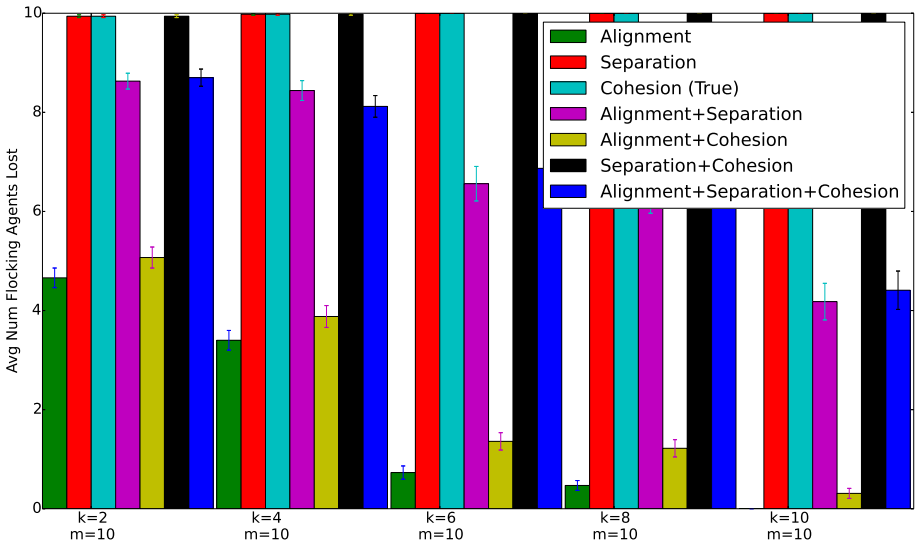


Figure 7.11: Results when *Cohesion* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

influencing agents behaving as if the flocking agents utilized the *Alignment* influence model to even save a few flocking agents from becoming lost.

Figure 7.13 shows results when the flocking agents are using the *Alignment* + *Cohesion* influence model. *Alignment* and *Cohesion* work well together to keep the flocking agents close to each other and oriented in the same direction. In Figure 7.13 we see that if the influencing agents assume the flocking agents are following either the *Alignment* influence model or the *true Alignment* + *Cohesion* influence model, then fewer flocking agents become lost. Performance is slightly better when the influencing agents assume the flocking agents are using the *Alignment* influence model because then the influencing agents are better able to orient the flocking agents towards the goal orientation θ^*.

Figure 7.14 shows results when the flocking agents are using the *Separation* + *Cohesion* influence model. *Separation* + *Cohesion* are conflicting forces, since *Separation* steers flocking agents away from neighbors while *Cohesion* steers flocking agents towards neighbors — but they are not exactly offsetting. In Figure 7.14, we see that performance is best for $k \geq 4$ when the

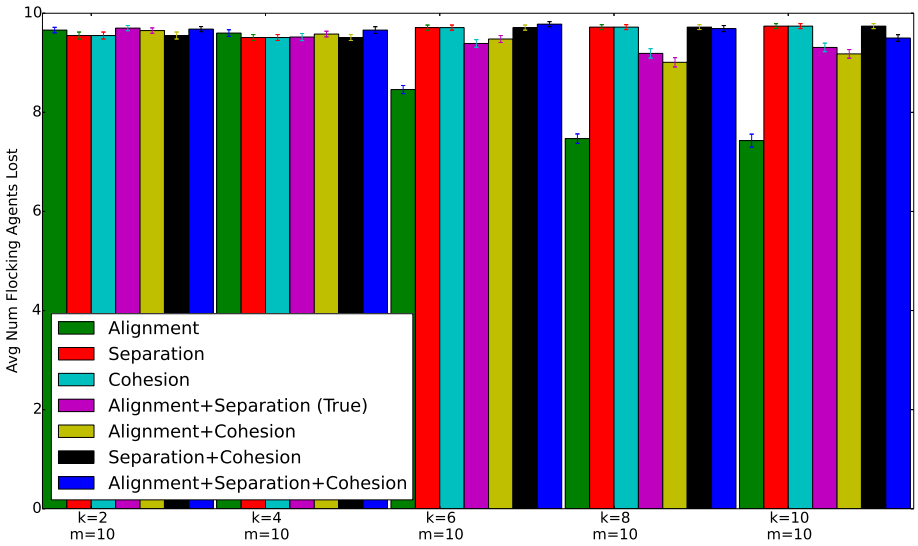


Figure 7.12: Results when *Alignment* + *Separation* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

influencing agents believe the flocking agents are utilizing the *Alignment* influence model. However, performance is second best when the influencing agents believe the flocking agents are utilizing the *Alignment* + *Separation* influence model. This is because influencing agents attempting to influence flocking agents using the *Alignment* + *Separation* influence model will influence the flock away from neighbors but in one direction.

Figure 7.15 shows results when the flocking agents are using the *Alignment* + *Separation* + *Cohesion* influence model. The *Alignment* + *Separation* + *Cohesion* influence model provides balanced flocking that avoids collisions while keeping the flock together both orientation-wise and location-wise. In Figure 7.15, we see that performance is best when the influencing agents believe the flocking agents are using the *Alignment* model. This is because the *Alignment* influence model helps the flocking agents stay together and orient towards the target orientation θ^*. The second best performance is obtained when the influencing agents believe the flocking agents are using the *true Alignment* + *Separation* + *Cohesion* influence model. This is not unexpected, since this model is (1) the model actually being used by the flocking agents and (2) a well balanced flocking model

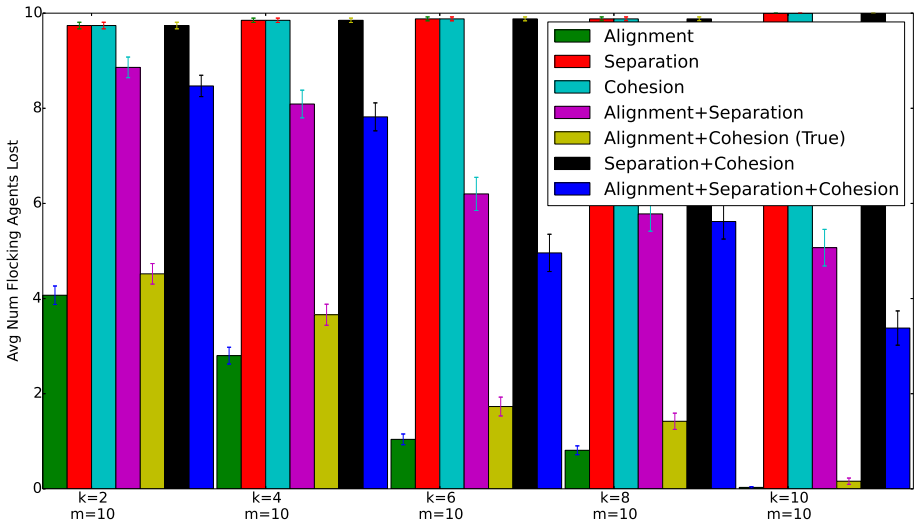


Figure 7.13: Results when *Alignment + Cohesion* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

that keeps the flock together in terms of orientation and location.

Throughout the figures in this section, we consistently found that it was best for the influencing agents to believe that the flocking agents are behaving according to the *Alignment* influence model. While this is partially an artifact of the *Alignment* influence model tending to result in fewer flocking agents becoming lost, it is also encouraging as it means that it may not be important for the influencing agents to know the exact influence model being utilized by flocking agents. In terms of our motivating example, this could mean that the robot birds that join a flock might not need to know the influence model of the flock they are joining in order to effectively influence the flock.

7.4 Summary

In this chapter, we set out to determine how well the algorithms and methods presented so far in this dissertation generalized to alternate neighborhood models and influence models. We also conducted experiments to determine whether it was critical for the influencing agents to know the

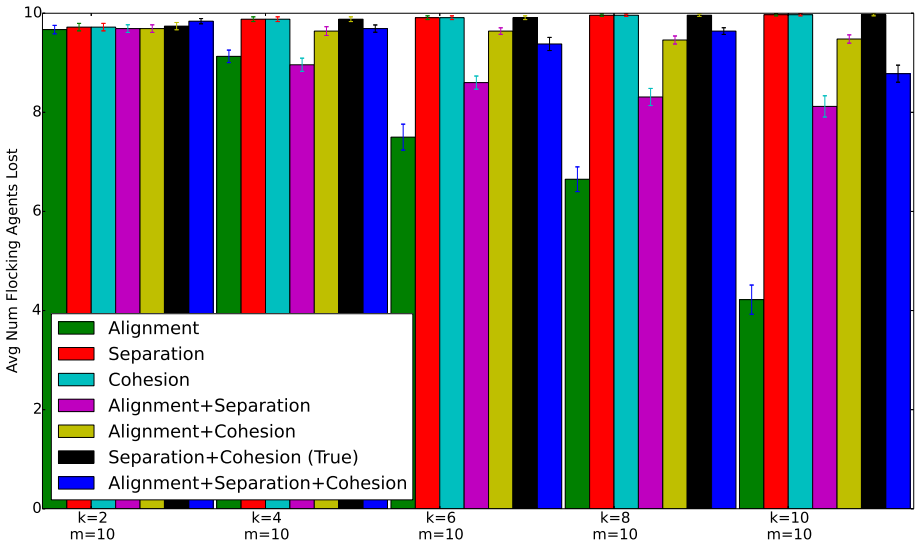


Figure 7.14: Results when *Separation + Cohesion* is the true influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

correct neighborhood model and influence model of the flocking agents.

In Section 7.2, we more deeply considered the alternate neighborhood models originally introduced in Section 2.1.1. In Sections 7.2.1, 7.2.2, and 7.2.3, algorithms were provided for determining which agents lie within an agent's neighborhood for each neighborhood model. Section 7.2.4 presented and analyzed results for these alternate neighborhood models. In these experiments we found that (1) the algorithms and methods in this dissertation generalize to the alternate neighborhood methods we considered and (2) it is not necessarily important that the influencing agents know the exact neighborhood model of the flocking agents. Section 7.3 discussed the seven combinations of the three aspects of Reynolds' algorithm for flocking. Section 7.3.1 experimentally evaluated the performance of each combination as well as how each combination faired when the influencing agents did and did not know the *true* influence model being utilized by the flocking agents. These experiments showed us that it is usually best for the influencing agents to believe that the flocking agents are behaving according to the *Alignment* influence model. This means that influencing agents may not need to determine the influence model of the flocking agents.

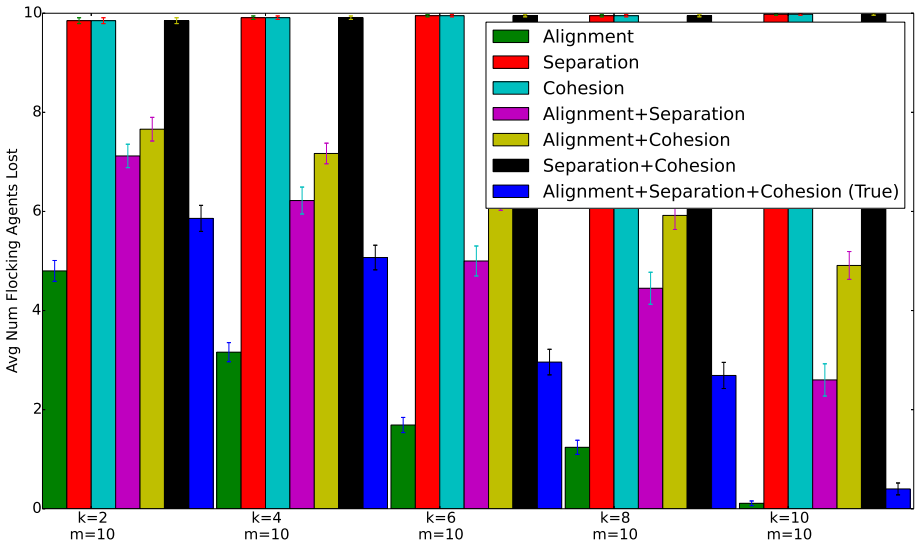


Figure 7.15: Results when *Alignment* + *Separation* + *Cohesion* is the *true* influence model used by the flocking agents. For these experiments, the influencing agents are originally positioned using the *Grid* placement method from Section 5.2.2. These graphs show results averaged over 100 trials, where the error bars depict the standard error of the mean.

This chapter considered questions regarding generalizability that represent a core component of this dissertation. The answers to these questions were generally positive, showing that it was not important for the influencing agents to know the exact neighborhood model or influence model that the flocking agents were utilizing. Chapter 8 will describe our work implementing parts of this dissertation — influencing agent behaviors from Chapter 4 and joining behaviors from Chapter 6 — on a real robot platform.

8. Robot Implementation

All of the algorithms and methods presented so far in this dissertation were designed and evaluated in simulation. Although simulation represents the real world, simulation is not the real world. In particular, simulation experiments are not fully reflective of the real world because the real world involves robots that (1) sense and act noisily and (2) have limited capabilities. Throughout this dissertation, we have considered the motivating scenario of using influencing agents to reduce birdstrikes at airports by guiding flocks of birds around the airports. Previous chapters of this dissertation showed that our algorithms for behavior, placement, joining, and leaving can effectively influence flocks of birds in simulation. In this chapter, we show that the primary algorithm for behavior described in this dissertation can guide a flock of SoftBank Robotics NAO robots around a dangerous area. As such, this chapter is a proof-of-concept that the algorithms in this dissertation could be implemented on robot birds to guide a flock of birds around a dangerous area.

In this chapter, we consider how the *1-Step Lookahead* algorithm from Section 4.1 performs when being used by a NAO robot to influence a flock of NAO robots.[1] Section 8.1 introduces the experimental set-up including the environment, NAO robot specifications, and an overview of the codebase we utilize for these experiments. Section 8.2 discusses how the flocking agent behavior was implemented on the NAO robots and the experiments run with flocking agents. Section 8.3 describes how the *1-Step Lookahead* algorithm from Section 4.1 was implemented on the NAO robots and the experiments we ran using influencing agents to influence flocking agents. Finally, Section 8.4 concludes the chapter.

The research question addressed in this chapter is: *How can influencing agents influence a flock of bipedal robots to avoid a particular area?* Although the experiments in this chapter were performed using bipedal robots, many of the lessons learned in these experiments would be applicable if extending the work in this dissertation to robot birds as discussed in Section 10.2.6.

[1] Videos of our experiments on NAO robots are available at `http://www.cs.utexas.edu/~katie/videos/`

8.1 Experimental Setup

Our lab has numerous NAO robots and an active RoboCup Standard Platform League (SPL) team, so we use the NAO platform and the UT Austin Villa[2] SPL codebase for our flocking experiments. In these experiments, our goal is to influence the flock to avoid a particular area of the environment.

8.1.1 Environment

The flocking experiments described in this chapter are held on the slightly-smaller-than-regulation 8 by 6 meter SPL field at the University of Texas at Austin. Our field is shown in Figure 8.1. The field has white lines and white goals that the robots use for localization.

Figure 8.1: Our flocking experiments are held on a SPL soccer field.

For the experiments in this chapter, our goal is to (1) influence the flock to avoid walking across the center spot at the center of the field by avoiding the field's center circle and (2) influence the flock to walk towards the soccer goal after walking around the center circle. We assume that each robot uses the *visibility radius* neighborhood model described in Section 2.1.1. In our experiments, we assume a visibility radius r of three meters.

[2] `https://www.cs.utexas.edu/~AustinVilla/`

8.1.2 NAO Robot

The SoftBank Robotics NAO robot is a 11.9 pound robot that stands 22.6 inches tall. NAO robots have 25 degrees of freedom (see Figure 8.2), two 1.22 megapixel cameras, left and right sonar sensors, an inertial unit with a 3-axis gyroscope and a 3-axis accelerometer, bump and force sensitive resistors on the feet, and three touch sensors on the head. A wireless network card and Intel Atom Z530 processor are built into the head. Each NAO uses a lithium-ion battery that allows for up to 60 minutes of active use.

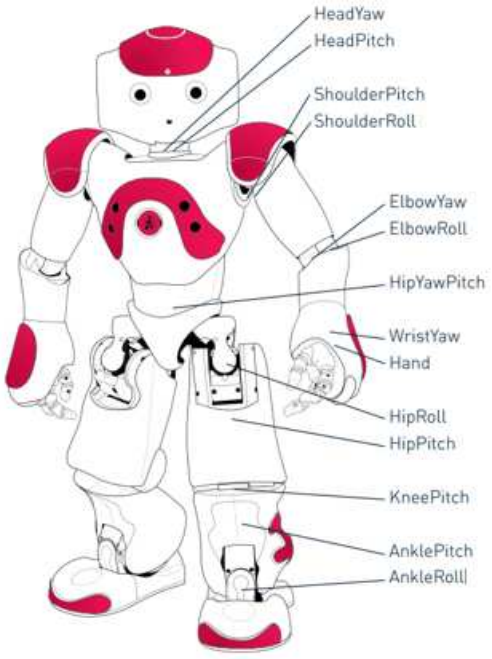


Figure 8.2: NAO robots have 25 degrees of freedom. Each joint is labelled with Joint Name[Motor Type][Reductor Type].[3]

NAO robots are designed for educational, entertainment, and customer-facing tasks. NAO is a rather affordable robot platform with a retail price in the United States of $9500/robot.

NAO robots come with an entire software library that allows users to quickly create animations and programs for the NAO. Among other things, this library allows users to easily get their NAO to move, recognize speech, recognize items, and track sound, faces, and colored items. However, this library is not well-suited for robot soccer research because the walking and behaviors are too slow. Although speed is not an issue for the flocking experiments described in this dissertation,

[3]By Aldebaran Robotics, `http://doc.aldebaran.com/2-1/family/nao_h25/motors_h25.html`

we use the UT Austin Villa codebase for our experiments because it is familiar to us and handles our vision, localization, and communication needs. The UT Austin Villa codebase is described in Section 8.1.3.

8.1.3 UT Austin Villa Codebase

The UT Austin Villa SPL codebase was used for the experiments in this chapter. A code release of the UT Austin Villa SPL codebase used at RoboCup 2016 is available on GitHub.[4]

The UT Austin Villa codebase is designed to support a team of five NAO robots playing soccer autonomously. As such, the codebase contains a vision module to allow the robots to make sense of the videos obtained by their cameras. Specifically, the vision module reports detections of field landmarks (such as goals, the center circle), other robots, and the ball. The localization module allows the robots to determine their position and orientation on the field based on both observed landmarks and their own motion. The motion module contains a kicking engine and walking engine. The walking engine feeds information regarding turning and walking to the localization module. Finally, the communication module handles inter-team communication such as location sharing, ball sharing, and role bidding. The most recent information received from each teammate is held in the *team packet* memory block.

Within our codebase, motion and high level strategy processes run 100 times per second. Vision processes run in a different thread 30 times per second.

8.1.4 Videos

Videos of our experiments on NAO robots are available on our website[5] but we also provide direct links to videos throughout this chapter as appropriate.

For each episode, we include a video of robot behavior. When available,[6] we also provide a video depicting each robot's real-time localization beliefs. Observing this localization information can explain some decisions by the robots that otherwise seem strange when watching behavior on the field. Do not assume that there is a significance to the color of each robot in the localization

[4] `https://github.com/LARG/spl-release`

[5] `http://www.cs.utexas.edu/~katie/videos/`

[6] Localization videos are not available for some episodes because the camera failed to record properly.

videos. There is significance in some cases though — in these cases, the significance is noted on our website as well as on the individual video pages.

8.2 Flocking Agents

Flocking agents comprise the flock that we wish to influence. In our experiments, the flocking agents always wear white jerseys while agents acting as influencing agents wear orange jerseys. In Section 8.2.1 we describe the behavior and implementation details of the flocking agents. In Section 8.2.2 we discuss experiments using flocking agents, while in Section 8.2.3 we discuss experiments in which a manually controlled flocking agent acts as an influencing agent.

8.2.1 Behavior and Implementation

For our robot experiments, we implemented the simplified version of Reynolds' flocking algorithm [68] that we utilized throughout most of this dissertation. As described in Section 2.1.2, this simplified version only considers the *alignment* aspect of Reynolds' flocking algorithm. Algorithm 14 shows pseudocode for flocking according to the *alignment* aspect of Reynolds' flocking algorithm. Table 8.1 introduces notation and variables that are utilized in Algorithm 14.

Variable	**Definition**
team_packets	A memory block containing the most recently communicated data from each teammate
team_packets.get(x)	Returns the data last received from teammate x
self	The robot's own teammate index
team_packets.get(x).locData	A localization memory block containing robot x's self-reported orientation (*orientation()*)
getTrueAngleDiff(neighOrient, selfOrient)	Returns the true angular difference between *neighOrient* and *selfOrient*

Table 8.1: Notation and variables used in Algorithm 14.

Algorithm 14 calculates the difference in orientation between a robot and the average orientation of its neighbors on lines 4–8. If the robot has no neighbors (line 9) or the difference in orientation is small (line 13), the robot walks forward (lines 10 and 14). Otherwise, the robot turns towards the average orientation of its neighbors (line 16). *turnThreshold* on line 12 can be tuned to

Algorithm 14 SimplifiedReynolds()

```
1: diffFromSelfTheta ← 0.0
2: numNeighbors ← 0
3: selfOrient ← team_packets.get(self).locData.orientation()
4: for each neighbor i index including self do
5:    neighOrient ← team_packets.get(i).locData.orientation()
6:    diffFromSelfTheta ← diffFromSelfTheta + getTrueAngleDiff(neighOrient, selfOrient)
7:    numNeighbors ← numNeighbors + 1
8: diffFromSelfTheta ← diffFromSelfTheta / numNeighbors
9: if numNeighbors == 0 then
10:   walk straight forward
11: else
12:   turnThreshold ← π/10
13:   if |diffFromSelfTheta| < turnThreshold then
14:      walk straight forward
15:   else
16:      turn in place to decrease diffFromSelfTheta
```

determine how different a robot's orientation must be from it's neighbors before it realigns. Smaller values of *turnThreshold* result in sensitive flocking agents that realign when their orientation differs slightly from that of their neighbors. On the other hand, larger values of *turnTheshold* result in flocks that can drift apart because the flocking agents do not realign until their orientation greatly differs from that of their neighbors.

8.2.2 Experiments with Flocking Agents

Although the work in this dissertation considered how to influence flocks from within, our initial robot experiments consider only flocking agents in order to ensure that the flocking behavior is correct. In this chapter, our goal is to prevent flocking agents from crossing over the center spot in the center of the field by influencing the flock to instead travel around the center circle. In this section we consider flocks of flocking agents moving downfield from the initial positions shown in Figure 8.3.

First, we considered two episodes in which robots flocked downfield. In the first episode, two robots flocked downfield but drifted slightly to the right while flocking.[7] In the second episode, three robots flocked downfield but drifted more significantly to the right while flocking.[8] While

[7]Robot video: `https://youtu.be/1G0iBRnSq00`, Localization video: `https://youtu.be/I0baV4Myx6o`
[8]Robot video: `https://youtu.be/z67dlB9O7Xs`

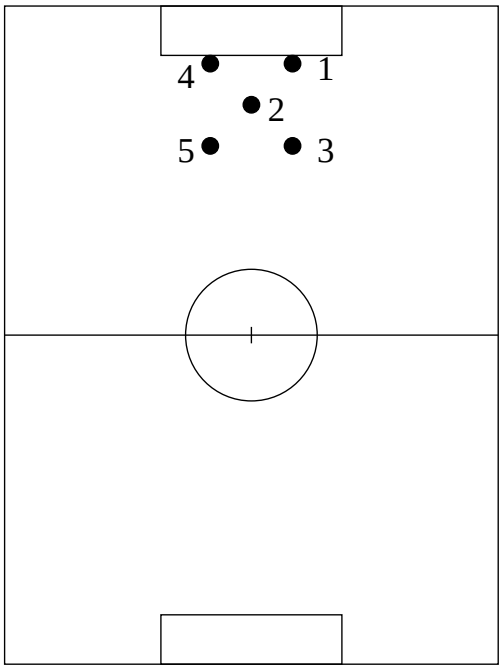


Figure 8.3: Initial positions of robots for experiments. Five possible positions are shown, although in some experiments not all five robots are utilized. Each position is labeled by the robot number, which corresponds to the numbers on the robot jerseys.

working on the NAO robots, we noticed that the robots sometimes do not walk completely straight when trying to walk "straight" — so the periodic, unpredictable drift observed in these episodes is likely a result of use patterns on the robots and not an issue with the flocking behavior.

Next, we considered an episode in which three robots flocked downfield while a fourth robots walked in to join the flock.[9] In this episode, the three robots turned slightly towards the right side of the field once the fourth robot entered their neighborhood. Likewise, the fourth robot began walking mainly towards the goal once the three robots entered its neighborhood. These behaviors by all four flocking agents were expected as a result of averaging their headings when they become neighbors.

8.2.3 Experiments Manually Influencing the Flock

The episodes described in Section 8.2.2 show that the flocking agents behave as expected. In this section we continue to run Algorithm 14's flocking agent behavior on all robots — but now we

[9]Robot video: `https://youtu.be/JZvahghiwEY`

physically control one robot in each episode to determine whether changes in one flocking agent's behavior can influence the flock to avoid a particular area of the environment. In other words, we manually re-orient one robot periodically to make it behave as we would expect an influencing agent would behave. For the remainder of this section, we will refer to the manually operated robot as an influencing agent.

In this section, we consider three episodes. There were multiple flocking agents and one influencing agent in each episode. During each episode, I manually turned the influencing agent when I believed doing so would influence the flock to travel around the center circle but not leave the field. Experience taught me that large, infrequent turns were most effective.

The first episode considered two flocking agents and one influencing agent.[10] In this episode, the flocking agents generally followed the influencing agent. There was some oscillation in orientations shortly after the influencing agent was turned, but the agents always converged to a general orientation. The flock became disoriented after the two-minute mark, but the flock eventually converged to continue towards the soccer goal.

The second episode considered three flocking agents and one influencing agent.[11] This episode progressed smoothly as the individual robots re-oriented when their orientation differed too much from the flock's orientation. Player 4 became lost at 1:45 in the robot video when the influencing agent was turned. Although player 4 recovered after making a complete turn, this behavior caused the flocking agents to converge to a different heading than the influencing agent at the end of the episode. Interestingly, the localization video showed that the flock believed it was converged to a single heading at the end of the episode.

The third episode considered four flocking agents and one influencing agent.[12] The influencing agent incrementally influenced the flock in this episode. However, after the flock made its way around the center circle, the influencing agent turned itself to the right at 1:26 in the robot video (although the localization video showed the influencing agent still flocking with the flock). This difference between the influencing agent's localization belief and reality caused subsequent turns of the influencing agent to behave unexpectedly. Despite these issues with the influencing agent,

[10]Robot video: `https://youtu.be/i_BaN4wesQ8`, Localization video: `https://youtu.be/Q0cmIYsJPJI`
[11]Robot video: `https://youtu.be/eIbRfM-8_QE`, Localization video: `https://youtu.be/FbdW0pqfCMk`
[12]Robot video: `https://youtu.be/KkvomKijJA4`, Localization video: `https://youtu.be/DJMXiDh5Ug4`

all of the robots except player 3 eventually converged to one heading. Although player 3 did not converge, the localization video showed that player 3 believed it converged with the flock.

There are some interesting trends in these episodes. First, the influencing agent usually turned back to its original orientation after being turned — and sometimes overshot it's original orientation — in flocks with fewer flocking agents. However, in the flock with four flocking agents, we never saw the influencing agent overshoot. The influencing agent likely did not overshoot when there were more flocking agents because the "pull" to the flock's new orientation was too strong. Second, turning the robot by large angles worked better than smaller angles, likely as a result of the overshooting just discussed. Third, turning the robot deliberately and placing it carefully was important — swinging the robot or turning it too quickly tended to result in the robot becoming lost. The robot likely became lost because the visual observations can become blurred if the robot is turned quickly. All of these observations helped guide our influencing agent behavior and implementation in Section 8.3.1.

8.3 Influencing Agent

This dissertation is about using influencing agents to influence flocks towards a particular behavior. Although the experiments in Section 8.2.3 considered how a flocking agent could be manually controlled to influence a flock, in this section we consider how influencing agents can influence a flock autonomously.

As in previous experiments, the flocking agents wear white jerseys while the influencing agents wear orange jerseys. In Section 8.3.1 we describe the behavior and implementation details for influencing agents. In Section 8.3.2 we discuss our experiments utilizing one influencing agent to influence a flock.

8.3.1 Behavior and Implementation

We implemented the *1-Step Lookahead* behavior from Section 4.1's Algorithm 5 as the influencing agent behavior for the experiments in this chapter. Since simulation is often different than reality, we made a few changes to Algorithm 5 as described below.

Algorithm 5 returns an orientation for the influencing agent to immediately adopt — yet robots cannot immediately adopt an orientation. As such, we altered Algorithm 5 such that it only considers — and hence can only return — orientations that the influencing agent can adopt within a few steps.

Additionally, we updated the *1-Step Lookahead* behavior for the influencing agents to either walk straight or walk forward while turning. The influencing agents walk straight while they have no neighbors. If the true angular difference between an influencing agent's current goal for the flock and the average heading of the flock is greater than $\frac{\pi}{6}$, then the influencing agent turns according to the *1-Step Lookahead* behavior to influence the flock towards the influencing agent's current goal for the flock. Otherwise, the influencing agent flocks with its neighbors by following Algorithm 14.

Finally, similarly to the maneuver experiments in Section 4.5, we assume each influencing agent always has a goal for the flock. In particular, the influencing agent's goal for the flock is always based upon the position on the field of the influencing agent's neighbors. For the experiments in this chapter, the influencing agents generally led the flock along the path shown in Figure 8.4. However, regardless of this path, the influencing agents would influence the flock downfield but towards the center of the field if any neighbors were close to leaving the left sideline of the field. Likewise, the influencing agents would also influence the flock towards the left sideline if any neighbors were close to entering the center circle.

8.3.2 Experiments with Influencing Agents

In this section we describe three episodes in which an influencing agent influenced flocking agents to avoid crossing over the center spot on a soccer field by influencing them to travel around the soccer field's center circle. In each of these episodes, the agents moved downfield from the initial positions shown in Figure 8.3.

In the first episode, the influencing agent successfully influenced two flocking agents to avoid the center circle.[13] The influencing agent first influenced the flock to turn and walk towards the sideline. Once player 5 was close enough to the sideline, the influencing agent influenced the flock to walk downfield between the sideline and the center circle. Once the influencing agent was near

[13]Robot video: `https://youtu.be/iuQdcsfWsfs`, Localization video: `https://youtu.be/P3lBLZlWLDA`

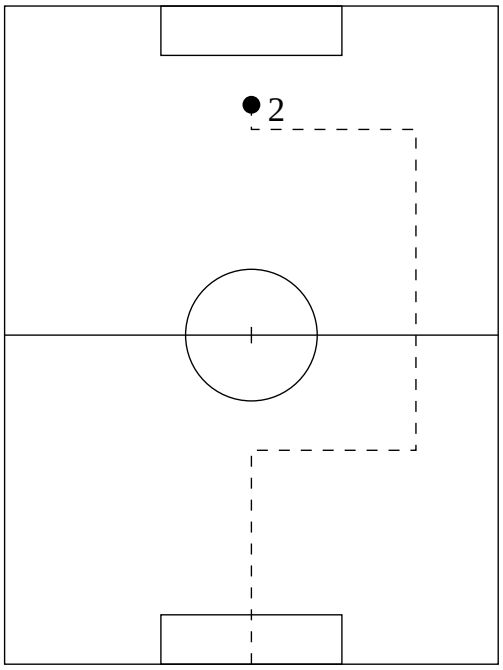


Figure 8.4: The influencing agent attempts to guide the flock along the dotted path shown in this figure.

the field's centerline, it slightly influenced the flock towards center field to keep player 3 from leaving the field. Once the flock walked past the center circle, the influencing agent influenced the flock to walk away from the left sideline. Once the flock neared the penalty cross, the influencing agent influenced the flock to turn towards the soccer goal. Note that throughout this episode, the localization video generally mirrored the robot video. Since the two videos depict similar behavior, we know that the robots remained well localized — or in other words, generally had accurate models of their current location and orientation.

As in the first episode, in the second episode the influencing agent influenced two flocking agents to avoid the center circle.[14] Although the robots remained well localized in the first episode, in this second episode the influencing agent became and then remained lost. At 2:18 in the robot video the influencing agent turned to the right, which influenced the flock to turn right towards the center circle. The influencing agent started to correct its behavior around 2:26 in the robot video, but continued to turn away from the soccer goal from 2:33 onward in the robot video. The robot video ended as the influencing agent walked off the field — yet surprisingly this did not

[14]Robot video: `https://youtu.be/3hLD1jL75z4`, Localization video: `https://youtu.be/Tkc4Kntr9J8`

influence the flocking agents negatively. This strange behavior in the robot video is explained by the localization video though. The influencing agent became lost after getting an unexpected line detection at 2:20 in the localization video. Then, although the influencing agent believed it was facing the goal from 2:32 onward in the localization video, a goal observation was not obtained until 2:55 in the localization video. The UT Austin Villa codebase does not use false negative data though, so the lack of a goal observation when expected did not affect the influencing agent's location and orientation beliefs. Unfortunately, the goal observations obtained from 2:55 onward in the localization video did not help the influencing agent though, since they were false detections of a goal near the field's sideline. These false detections led to the influencing agent walking off the right sideline while it believed it was walking off the endline.

In the third episode, the influencing agent influenced four flocking agents to avoid the center circle.[15] The influencing agent initially turned to influence the flock towards the left sideline. The influencing agent continued attempting to influence the flock to turn further away from the center circle until 0:40 in the robot video when it converged with the flock towards the left sideline. Note that players 1 and 4 became lost at 0:52 in the robot video, but corrected their behavior by 1:00 in the robot video. At 1:05 in the robot video, player 2 began influencing the flock upfield. At 1:53 in the robot video, the influencing agent turned to leave the left side of the field in order to influence the flock away from the center circle because player 1 had walked too close to the center circle. Although the influencing agent fell, the influence was effective enough to keep player 1 from entering the center circle. The influencing agent tried at 2:14 in the robot video and 2:34 in the localization video to influence the flock to turn left towards the soccer goal. However, the influencing agent had no room to walk due to the edge of the field and obstacles lying just off the field. Meanwhile, the flock walked farther from the influencing agent — which now had multiple localization models as shown in the localization video at 2:41 —and no longer considered the influencing agent as a neighbor. Note that player 4 broke near the end of the episode and was removed from the field. When comparing this third episode to the first and second episodes, the influencing agent needed to turn much more to influence a flock of four robots than a flock of two robots. These substantial turns required the influencing agent to behave differently than the flock for long periods of time.

[15]Robot video: `https://youtu.be/g8iIKV1YrCk`, Localization video: `https://youtu.be/EWuWIEAn2X0`

As happened in the third episode, behaving different for long periods of time can result in the flock leaving the influencing agent's influence.

These three episodes show that using an autonomous influencing agent to influence a flock to travel around a dangerous area is possible. Influence over the flock could be improved in a few ways though. First, it would be beneficial if the influencing agent was able to keep the flock more cohesive and hence maintain influence over all of the flocking agents throughout the experiment. Second, improving localization information would result in more accurate influence by the influencing agents. Finally, designing algorithms that allow the influencing agents to trade-off between influencing and staying within the neighborhoods of flocking agents may be beneficial.

8.4 Summary

In this chapter, we considered how the *alignment* aspect of Reynolds flocking algorithm and the *1-Step Lookahead* algorithm from Section 4.1 perform when being used in flocking experiments on NAO robots.

In particular, Section 8.1 introduced the environment, NAO robot platform, and codebase that we used in our robot flocking experiments. Section 8.2 discussed how the *alignment* aspect of Reynolds flocking algorithm performed on NAO robots. Section 8.2 also presented results for manually operating one flocking agent to behave as an influencing agent. Finally, Section 8.3 described how the *1-Step Lookahead* behavior was altered to be used to influence a flock of NAO robots along a path. Three episodes using the *1-Step Lookahead* algorithm on one influencing agent to influence a flock were discussed.

This chapter shows that the primary influencing agent behavior presented in this dissertation — the *1-Step Lookahead* algorithm from Section 4.1 — could be utilized successfully in a robotic domain with minimal alterations to account for the difference between simulation and reality. The success of the *1-Step Lookahead* algorithm in this chapter's experiments provides evidence that the algorithms in this dissertation could be used on robot birds to influence flocks of birds to travel around dangerous areas, such as windfarms and airports.

9. Related Work

Our work towards influencing the behavior of flocks is inherently motivated and influenced by a variety of cross-disciplinary fields. As such, this chapter overviews some of the related and relevant work in these cross-disciplinary areas.

In this chapter, we review related work in the multiagent coordination and teamwork sector in Section 9.1. In Section 9.2 we consider related work on *ad hoc teamwork*, or teamwork that is not pre-coordinated. In Section 9.3 we consider relevant and motivational flocking, herding, and swarm research. We include subsections detailing related work on the two most common flocking formations: cluster formations (Section 9.3.1) and line formations (Section 9.3.2). Finally, we consider work by others that is most similar to our own — work on influencing a flock — in Section 9.4. In particular, we consider three main methods for influencing a flock in Section 9.4: human-led influence (Section 9.4.1), shepherding (Section 9.4.2), and infiltration (Section 9.4.3).

Throughout this chapter, for each piece of related work we describe how the work in this dissertation is either (1) different or (2) addresses a related but distinctly different problem.

9.1 Multiagent Coordination and Teamwork

Multiagent teams, both in industry and in academia, are almost always explicitly designed to coordinate. Agents on these teams are usually designed specifically to work with other agents on these teams such that their behaviors are tightly coupled.

Most multiagent teams require explicit coordination protocols or communication protocols. Three popular protocols for communication and coordination — SharedPlans [40], Shell for TEAMwork (STEAM) [76], and Generalized Partial Global Planning (GPGP) [24] — all provide collaborative planning or teamwork models to each team member. Each of these protocols work well when all agents know and follow the protocol. However, in this dissertation we do not assume that any protocol is known by all agents and hence we cannot successfully use such protocols in

our work.

Some multiagent teams are designed to work specifically with their teammates in pre-defined ways. Stone and Veloso introduce the idea of periodic team synchronization domains [72]. In these domains, a "locker-room agreement" is formed and the team uses periodic team synchronization periods to coordinate their teamwork structure and communication protocols. The work discussed in this dissertation differs from this work in that we do not assume the availability of a team synchronization (pre-coordination) period.

Yu *et al.* propose an implicit leadership algorithm that allows all agents to follow a single simple rule and effectively reach a common group decision without any complex coordination methods [82]. Specifically, implicit leadership allows all agents to agree on a decision that can be determined by one or a few informed agents. Their approach can also handle cases where informed agents have different confidence levels regarding their information. Under their model, each agent is able to control particular state variables and determine the state of neighbors within a particular radius. Each informed agent has a goal state and a confidence in that goal. According to their model, each agent attempts to align its state with those of its neighbors and informed agents also attempt to achieve their goal state. The work by Yu *et al.* uses different methods to address the same problem that is considered by Chapter 4. However, since their control law incorporates both alignment with neighbors and alignment to a goal state, the group as a whole will reach the goal state slower using Yu *et al.* 's algorithm than using our *1-Step Lookahead* behavior shown in Algorithm 5.

One instance of large-scale multirobot coordination is Amazon Robotics' Kiva system [22, 27]. The Kiva system involves hundreds of mobile robots working together in large warehouses to deliver inventory pods to a variety of locations. Some of these deliveries are time sensitive, such as delivering the pods to and from packers. Other deliveries can be done at any time, such as moving out-of-season pods to more remote sections of the warehouse. Although most of the details of the Kiva system are unknown by those outside the company, it is clear that this is a real-life large-scale multi-agent resource allocation problem.

In addition to the work described in this dissertation, I have also worked extensively on the UT Austin Villa RoboCup Standard Platform League (SPL) team [9, 10, 11, 33, 63]. For our

SPL team, we program five SoftBank Robotics NAO robots to play soccer together. Our robots communicate using a specific communication protocol that allows them to explicitly coordinate the position of each teammate, the assignment of roles to each teammate, and the current play selection. Even if a robot designed outside our team were able to use our communication protocol, our behaviors are written in such a way that we assume particular characteristics, such as walk speed and what it means to play a particular role, that it would be difficult for a robot designed outside of our team to naturally fit into our team. Interestingly, we will get to test out this exact situation at RoboCup 2017 when we compete in the new *mixed team tournament*.[1] In this new competition, we will compete on a joint team with the UPennalizers[2] from the University of Pennsylvania, where the goal is for both teams to use a shared communication protocol to play together as a pre-coordinated team despite utilizing independent code bases.

9.2 Ad Hoc Teamwork

Although coordinated teamwork is a well-studied area, most research addresses the problem of coordinating and communicating among pre-coordinated teams that are designed to work together. Ad hoc teamwork, on the other hand, addresses multiagent teamwork in which the coordinating agents do not all share a common coordination framework. The work in this dissertation is motivated by ad hoc teamwork — hence in this section, we overview recent work under the ad hoc teamwork umbrella. Each of these projects is related to this dissertation in that they each consider coordination without pre-coordination when one or more agents join a team. However, each project differs in (1) how the ad hoc agents attempt to work with the team they are joining and (2) the dynamics of the team they are joining.

In a 2010 AAAI challenge paper, Stone *et al.* challenged the artificial intelligence community to develop agents that are able to join previously unfamiliar teammates to complete cooperative activities [71]. Although they were not the first to consider this problem — two earlier works [15, 51] are discussed below — they did draw attention to this under-researched part of multiagent

[1] Mixed team tournament rules (Appendix B):
`http://spl.robocup.org/wp-content/uploads/downloads/Rules2017.pdf`

[2] `https://fling.seas.upenn.edu/~robocup/wiki/`

systems and they coined the terminology "ad hoc teamwork" and "ad hoc agents" to describe work in this area. Their paper provided a definition of ad hoc teamwork, a methodology for evaluating performance of various ad hoc agents when paired with various teammates in a particular domain, and an initial assessment of the potential technical challenges that should be addressed when creating an ad hoc agent.

In the robot soccer domain, Bowling and McCracken [15] propose methods for coordinating an agent that joins an unknown, pre-existing team. In their work, each ad hoc agent is given a playbook that differs from the playbook of its teammates. The teammates assign the ad hoc agent a role, and then react to it as they would any other teammate. The ad hoc agent analyzes which plays work best over hundreds of simulated games, predicts the roles its teammates will adopt in new plays, and assigns itself a complementary role in these new plays. Similarly to the work in this dissertation, they propose coordination techniques for an agent that wants to join a previously unknown team of existing agents. However, they take a different approach to the problem in that they provide the single agent with a play book from which it selects the play most similar to the current behaviors of its teammates.

Jones *et al.* perform an empirical study of dynamically formed teams of heterogeneous robots in a multirobot treasure hunt domain [51]. In their work, they adapted the Traderbots system [26] to dynamically form heterogeneous teams. They assumed that all of the robots know they are working as a team and that all of the robots can communicate with one another, whereas we do not assume that the teammates realize they are working on a team with the ad hoc agents nor do all of the agents share a communication protocol.

In 2013, Liemhetcharat defended a dissertation in which he considered (1) how to model how well teammates work together on an ad hoc team, (2) how to learn such models, and (3) how to use this knowledge to form more effective ad hoc teams [57]. Liemhetcharat formally defined a weighted synergy graph that models the capabilities of robots in different roles and how different role assignments affect the overall team value. He presented a team formation algorithm that can approximate the optimal role assignment policy given a set of teammates to choose from and a task. He also used observations of a team's performance and attempted to fit models to this data, where the data could either be provided all at once (if previous observations are available) or online (to

update the model as observations are acquired). Liemhetcharat's work is similar to the work in this dissertation in that both consider adding ad hoc agents to teams. However, in this dissertation we consider how ad hoc agents should behave, whereas Liemhetcharat considers which ad hoc agents should be added to a team (given multiple possible options).

In 2014, Barrett defended a dissertation in which he considered how to use limited knowledge about teammates to plan how to best act [8]. Barrett focused on algorithms that allowed ad hoc agents to learn about their environment and teammates, as well as reason about teamwork and choose appropriate actions. He created ad hoc agents that were (1) robust to a variety of teammates by being able to learn about teammates and adapt to their behaviors online, (2) robust to a variety of tasks by being able to adapt to new tasks and decide when to take actions to learn about the behaviors of teammates, and (3) able to adapt quickly to new teammates and tasks without extensive observations of either. As such, Barrett created ad hoc agents that could work well, but not necessarily optimally, with a variety of unknown teammates on a variety of tasks. Our work differs from the work of Barrett in that we specifically focus on all aspects of creating ad hoc agents that can work well with teammates exhibiting a reactive swarm behavior.

In 2015, Albrecht defended a dissertation that considered how to design an agent that is able to achieve optimal flexibility and efficiency within a team despite having no prior coordination [2]. One of his main contributions is the Harsanyi-Bellman Ad Hoc Coordination algorithm which uses concepts from game theory to facilitate ad hoc agents coordination with previously unknown agents. As with Barrett's dissertation [8], our work tackles an inherently different problem than Albrecht's work. Specifically, we focus on creating a complete ad hoc agent that can work well with teammates in a reactive swarm.

Various ad hoc teamwork experiments have occurred at recent RoboCup competitions. A Drop-in Player Competition has been held in the 3D simulation league and Standard Platform League (SPL) since 2013 [32, 61]. We have been involved in organizing and running the SPL Drop-in Player Competition since it began in 2013. This competition pulls one player from five different teams, and puts these players on a team to play with no pre-coordination and limited communication. The RoboCup Small Size League has also held a "Mixed-team Tournament" [69] at some RoboCup competitions in which two teams are randomly combined to play as one. The

winner of their tournament is the team who wins the most games.

9.3 Flocks, Herds and Swarms

Many researchers in various fields, including biology, physics, graphics, and computer science, have considered the emergent behavior of flocks, herds, and swarms. In this section, we discuss some of the work that is most related and relevant to the work of this dissertation.

Barca formulated a template for an anti-swarming strategy that could be utilized against adversary robot swarms [7]. His stance is that since swarm robots will be widely available in the near future, it is critical to develop technologies to impede destructive swarms before they become problematic. The techniques described in his paper try to bring destructive swarms into a calm state, called recession. It is possible that the methods presented in this dissertation could be extended to influence dangerous swarms into a safe area (such as a large metal box). Towards this idea, Section 10.2.9 considers this exact application.

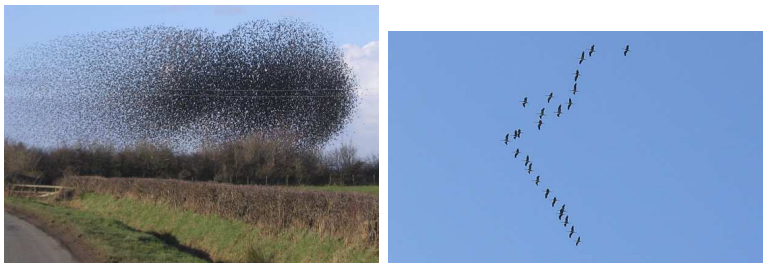
Cristiani and Piccoli introduce an agent-based model for simulation that takes into account only long-range cohesion, short-range repulsion, and visual field [20]. Using this simple model, they are able to generate many different self-organized patterns that cover the behaviors observed in nature for most animal groups by simply setting the parameters of their model. Although their work does not consider the effect influencing agents can have on these animal groups, it does imply that the algorithms and methods created for one model may be applicable to other models with simple parameter tuning.

Berger *et al.* present a general method for swarm classification from partial data using subspace learning [12]. They are able to use this method to classify swarm behavior as well as recognize emerging swarm behavior. Their results show that their method performs better than the state-of-the-art in swarm classification across a variety of swarm models and agent sampling schemes. Although Chapter 7 showed that knowing the exact flocking model is not crucial in most cases, the work by Berger *et al.* is relevant in the cases where being able to accurately classify swarm behavior — such as the flocking model being employed by a flock — could improve performance.

Bajec and Heppner organized a through review of organized flight in birds [5]. They noted

that the fraction of active investigators of organized flight in birds with a biology background had steadily decreased since the 1970s. Their claim was that continued progress in the area would depend on cross-disciplinary collaborations instead of specialists working alone. Their review focuses on two types of organized flight — cluster formations and line formations — and pays specific attention to the types of questions usually asked by researchers for both types of organized flight.

For the remainder of this section, we will consider the research related to cluster formations (Section 9.3.1) and line formations (Section 9.3.2) that is most relevant to the work presented in this dissertation. For clarity, Figure 9.1 shows an example of each type of formation.



(a) Cluster Formation (b) Line Formation

Figure 9.1: Pictorial descriptions of the two main types of organized flight.[3]

9.3.1 Cluster Formations

The flocking model that we introduced in Section 2.1 and assumed throughout this dissertation results in agents flocking in a cluster formation such as seen in Figure 9.1(a). Some biologists believe that a cluster formation is utilized by many types of small birds because the tight cluster offers protection against aerial predators by increasing the risk of collision to the predator [77]. Bajec and Heppner provide a complete review of flocking cluster formations [5]. In this section, we consider related work that concerns flocks utilizing a cluster formation.

Reynolds introduced the original algorithm for flocking that we use in this dissertation [68]. His work focused on creating a realistic computer model of flocking. As described in Sections 2.1.2 and 7.3, Reynolds' algorithm for flocking consists of three simple steering behaviors that determine how each agent maneuvers based on the behavior of the agents around it (henceforth

[3](a) By John Holmes, CC BY-SA 2.0, `https://commons.wikimedia.org/w/index.php?curid=9240013`
(b) By Andreas Trepte - CC BY-SA 2.5, `https://commons.wikimedia.org/w/index.php?curid=337516`

called *neighbors*): *Separation* steers the agent such that it avoids crowding its neighbors, *Alignment* steers the agent towards the average heading of its neighbors, and *Cohesion* steers the agent towards the average position of its neighbors. Vicsek *et al.* considered just the *Alignment* aspect of Reynolds' algorithm for flocking [80]. Hence, like in all of this dissertation except Section 7.3, Vicsek *et al.* use a model where all of the particles move at a constant velocity and adopt the average direction of the particles in their neighborhood. However, like Reynolds' work, Vicsek *et al.* were only concerned with simulating flock behavior and not with adding controllable agents to the flock.

Of course, Reynolds' algorithm for flocking is just an approximation of how actual flocking occurs. In reality, it is difficult or impossible to determine how individual birds in flocks decide how to behave. Herbert-Read *et al.* show that some of the commonly-held approximations and beliefs about how flocking in cluster formations occurs — such as Reynolds' flocking algorithm [68] — may not be consistent across different species [47]. Specifically, Herbert-Read *et al.* presented three key rules for the social interactions of mosquitofish (Gambusia affinis): (1) attraction forces are important in maintaining group cohesion, but there is only weak evidence that fish align with their neighbor's orientation, (2) repulsion is mediated principally by changes in speed, and (3) although the positions and directions of all flock members are highly correlated, individuals only respond to their single nearest neighbor. Some of these findings directly conflict with Reynolds' flocking algorithm [68] and our flocking model assumptions in Section 2.1. This difference could be a result of mosquitofish not following conventional flocking behavior — or this could be a sign that one flocking model cannot accurately describe flocking across different species. In either case, even if flocking models are inconsistent within a species, we found in Chapter 7 that many of our algorithms and methods can apply across different flocking models.

It is often difficult to identify the precise interaction rules used by different species. However, recent advancements in tracking technology now allow large amounts of data to be collected on the movements and positions of individuals in groups. Students and professors at Georgia Tech released the Biotrack software for tracking and analyzing multiple agents.[4] Biotrack has been used successfully to track ants, bees, and termites. The Biotrack software uses a tracking approach that is based on work by Feldman *et al.* [28]. Feldman *et al.* contribute a greedy detection-based

[4] `http://www.bio-tracking.org/category/software/`

algorithm for tracking dynamic targets in an online fashion. In experiments, Feldman *et al.* show that they can use four to eight laser range finders to track humans in sports and social settings with over 98% correct detections. In other animal tracking work, Herbert-Read discussed how trajectory data can be used to model how animals act in groups [46]. The work by both Feldman *et al.* and Herbert-Read is relevant to the work in this dissertation because creation of more accurate animal behavior models will be helpful in determining how to best influence these animals.

Some related research has also considered how different information provided to the flocking agents affects their behavior. Turgut *et al.* consider how noise in heading measurements, the number of neighbors, and the range of communication affect the self-organization of flocking robots [79]. Their experiments show that the range of communication is the primary factor that determines how many robots can flock together. They found that the flocking agents are highly robust to noise in heading measurements and neighbors. Moeslinger *et al.* present a flocking algorithm for low-end flocking robots that have no ability to communicate but do have four distance sensors with limited range [64]. However, neither of these research lines consider how to influence the flock to adopt a particular behavior by introducing additional agents into the flock.

Strandburg-Peshkin *et al.* consider the accuracy of commonly used neighborhood models in fish [73]. They present a novel approach that considers individual movement decisions to be based explicitly on the available sensory information. Specifically, their visual model claims that all individuals that occupy an angular area on the retina of the focal fish that is greater than a threshold value are considered to be neighbors. Explicitly considering visual sensing allows them to accurately predict the propagation of behavior changes in groups due to leadership. They claim that the structural properties of visual interaction networks differ markedly from commonly used metric and topological models, meaning that these models likely do not reflect the visual information employed when making movement decisions. In their experiments, they use some informed agents that are trained to move towards a stimulus. They note that influence can be seen to propagate through the flock like a wave and that informed individuals respond first and tend to occupy frontal positions in the group. Like Strandburg-Peshkin *et al.*, we have noticed that influence does tend to propagate through the flock like a wave. Although the similarities between this dissertation and their work do not extend past this observation, their work is relevant to our work in that it

considers the reality of various neighborhood models.

Zavlanos *et al.* present a theoretical framework for controlling graph connectivity in mobile robot networks and consider flocking as an application of connectivity control [83]. In their multi-robot system, they were able to guarantee flocking behavior while preserving network connectivity. Their work did not consider adding influencing agents to the flock, but it is notable because they consider network connectivity in flocking and this inspired the graph algorithm for influencing agent placement that is described in Section 5.3.

Rosenthal *et al.* show that the connectivity between schooling fish by which behavior propagates is complex, weighted, directed, and heterogeneous [70]. Specifically, they show that individuals with relatively few strongly connected neighbors are both most socially influential and most susceptible to social influence. This finding is contrary to the assumption made in our work that the agents respond to each other in a homogeneous way. Examining the impact of changing this assumption in our work is discussed as part of our suggested future work in Section 10.2.5.

9.3.2 Line Formations

Line formations — often exhibited as V-like formations, as seen in Figure 9.1(b) — behave very differently than the Reynolds' flocking algorithm we considered throughout this dissertation. As such, most of the algorithms and methods in this dissertation will not directly apply to influencing V-like formations. With this in mind, one of the areas of future work suggested in Section 10.2.7 is to consider how to influence a V-like flock formation. In this section, we consider related work that concerns flocks using a line formation.

Nathan and Barbosa describe a V-flocking model via a small set of positioning rules. These positioning rules allow the flock to stabilize into several well-known V-like formations that have been observed in nature [65]. These positioning rules are composed of (1) seek the proximity of the nearest bird (coalescing), (2) if coalescing does not apply, seek the nearest position that affords an unobstructed longitudinal view (gap-seeking), and (3) apply gap-seeking while the view that is sought is not obtained or the effort to keep up with the group decreases due to increased upwash (stationing). Wilkerson-Jerde *et al.* implemented these rules in NetLogo [81].

The V-flocking model described by Nathan and Barbosa [65] depends heavily on the concepts

of upwash and downwash. Portugal *et al.* examine upwash exploitation and downwash avoidance using real-life data of a free-flying flock of ibises [67]. In their experiments, back-mounted integrated Global Positioning System and inertial measurement units recored the position and every wing flap of 14 birds during 43 minutes of migratory flight. Their experiments show that the ibises were able to (1) either sense or predict the spatial wake structure of neighbors and (2) use strategies to cope with the dynamic wakes produced by the flapping wings of neighbors. Effectively, the birds would flap in sync if they were a full wavelength behind another bird and exactly out of sync if they were a half wavelength behind another bird. These abilities to sense, predict, and cope were previously not thought possible for birds due to the required flight dynamics and sensory feedback.

Klotsman and Tal introduce an alternate V-flocking model that combined a data-driven approach with an energy-savings model [55]. In their work, they consider both a "flock initiation" phase where the flock converges to a target shape as well as a "steady flight" phase during which energy-savings models are calculated and used. Specifically, during the "steady flight" phase they use learning from examples to determine the most energy-efficient parameters for "steady flight" flocking.

Andersson and Wallander consider why some flocks use line formations [3]. They discuss the energy saving aspects of line formations, but also consider the effect of kin selection and reciprocity. They believe that the bird leading an acute V formation saves less energy than the trailing birds, while the disadvantage of leading is reduced in more obtuse formations. Andersson and Wallander found that acute V formations occur mainly in circumstances conducive to kin selection or reciprocity, such as in small flocks of adults with offspring or small groups of unrelated individuals that take turns leaving the V formation. Likewise, obtuse V formations occur mainly among unrelated individuals, such as those migrating long distances.

Unlike our work, all of the work in this section studies flocking models — the dynamics of these V-flocking models or why these models are biologically feasible. None of the work in this section considers how influencing agents added to a V-like flock can influence the flock towards a particular behavior.

9.4 Influencing a Flock

Other researchers have considered how flocks can be influenced. The work most closely related to our work is discussed below, as well as how each piece of related work differs from the work in this dissertation.

We first consider work on human-led influence in Section 9.4.1. Then, in Section 9.4.2 we discuss work on shepherding, where shepherding is a flocking behavior in which outside agents attempt to control a flock via herding them. Finally, in Section 9.4.3 we discuss work that — like the work in this dissertation — utilizes one or more influencing agents. The work covered in Section 9.4.3 is the most related to the work in this dissertation as in this work, the agents that attempt to control the flock are seen by the rest of the flock as friendly, homogeneous flock mates.

9.4.1 Human-led Influence

Some related work considers how humans can influence flocks. This influence could take the form of controlling some subset of the flock, providing limited instructions to the flock as a whole, etc. In this section, we discuss the work on human-influenced flocks that is most relevant to the work described in this dissertation.

Tiwari *et al.* consider the problem of effectively steering a large robot swarm [78]. They assume that each individual robot can either be controlled by a human or behave according to a zonal self-propelled particle swarm model. Although we did not consider this zonal model in Section 2.1, we do suggest it as future work in Section 10.2.5. Tiwari *et al.* consider which robots should be controlled by a limited number of humans. Specifically, they consider whether the robots controlled by humans should be located at the (1) front, (2) middle, or (3) periphery of the flock. Although Tiwari *et al.* consider which locations are best for influencing a swarm, their work differs from the work of this dissertation in that they do not consider how the controlled agents should behave (because they are controlled by humans) nor do they consider how the controlled agents should join or leave the swarm (since the controlled agents are already part of the swarm).

Kerman *et al.* propose a bio-inspired swarm model and show that the model has two fundamental attractors: a torus attractor and a flock attractor [53]. Two metrics of group behavior

are used to define the two attractors: group angular momentum and group polarization. Their work studies the stability of both of these attractors and shows that a control input can be used to force the swarm to change from one attractor to the other. Additionally, their work shows how a human operator can encourage the swarm to change from one attractor to another by partially controlling some subset of the swarm. Brown *et al.* build upon the work of Kerman *et al.* [53] by considering how human interaction with robot swarms can be handled at a high level of abstraction [17]. This allows the human to abstract the details of individual agents and instead focus on managing the swarm as a whole. Their work shows that only limited human influence is needed to cause the swarm to switch between a flock and a torus. In their work, switching between a flock and a torus is easier for the human to encourage than in [53] because they found a set of parameter values at which both behaviors could be observed if a subset of the swarm was perturbed enough. Although [53] and [17] focus on a different problem than the work in this dissertation, it is encouraging that partial control over a subset of the swarm is able to influence the swarm to switch between attractors.

Jung *et al.* present a new shared control method for human-swarm influence that uses *mediators* that operate from within the spacial center of the swarm [52]. Two types of mediators are considered: a *repulsion mediator* that repels neighbors similar to a predator and a *repulsion and attraction mediator* that attracts neighbors but also includes a repulsion region within the attraction region. A human operator can use these *mediators* to transform and move a dynamic torus formation while sustaining influence over the torus, avoiding fragmentation, and maintaining the torus' connectivity. Although this work focuses on human-controlled agents, the work in this dissertation could be extended to consider how similar mediators could infiltrate a flock in order to influence the flock to behave in a particular manner.

9.4.2 Shepherding

Groups of agents can be influenced by shepherding behaviors. Shepherding behaviors are a type of flocking behavior in which outside agents (shepherds) attempt to control the motion of another group of agents by exerting repulsive forces on them. Shepherding is often also referred to as "herding." All of the shepherding work discussed in this section differs from the work in this dissertation

because it considers behaviors for shepherds that repel the flock while our work considers behaviors for influencing agents that are seen by the flock as friendly, homogeneous flock mates.

From 2008 to 2010, the ProMAS workshop at AAMAS held a competition[5] in which teams competed against each other in a grid-like world in which simulated cows are moving collectively in multiple groups showing swarm-like behavior. Each team's goal was to herd as many cows into its corral as possible. However, herding a flock into a corral is concerned with short-term behavior of the flock whereas attempting to influence a flock to adopt a particular flocking behavior is concerned with more long-term flock behavior.

King *et al.* use global positioning system data to study the response of a group of sheep to an approaching sheepdog [54]. Analysis of movement trajectories shows that the sheep exhibited a strong attraction towards the center of the flock while being threatened by the sheepdog. This data supports the common belief that individuals respond to threat by moving towards the center of the threatened group. With regards to our work, the experimental data from King *et al.* provides evidence that guiding flocks away from dangerous areas using the methods described in this dissertation may be more effective and predictable than attempting to scare the flocks away using real or robot predators.

Lien *et al.* present a variety of shepherding behaviors that use one or more shepherds to control the flock [58, 59]. In [58], three different approaching methods and three different steering behaviors are considered for four types of shepherding behaviors. Only one shepherd is considered in [58], but Lien *et al.* extend their work in [59] to consider multiple shepherds. In [59], two shepherd formations and three methods for matching shepherds with steering points are considered. The approaching and steering methods in [58], as well as the shepherd formations in [59], are relatively simple. Specifically, the "Using a Straight Line" approaching behavior in [58] and the "Line Formation" and "Arc Formation" shepherd formation in [59] are similar to the *target formations* considered in Section 6.1.2 of this dissertation.

Pierson and Schwager introduce a control strategy for non-cooperative herding in which dog-like robots attempt to herd sheep-like agents to a specific point [66]. Strombom *et al.* present a general algorithm for shepherding that is based on adaptive switching between collecting dis-

[5] `https://multiagentcontest.org/`

persed agents and driving aggregated agents [74]. Extensive experimental results show that (1) this algorithm can herd the flock towards a target destination and (2) the resulting behavior is consistent with real shepherding events involving sheep and sheep dog. However, like in Lien *et al.* [58, 59], the work in both Pierson and Schwager and Strombom *et al.* differs from our own in that our influencing agents are seen by the flock as homogeneous flock mates while in this work the shepherds repel the sheep.

Harrison *et al.* present a motion planning strategy for shepherding in environments with obstacles [43]. Their strategy allows shepherds to view the flock as a discretized deformable shape. Viewing the flock as a deformable shape makes the strategy applicable to robotics scenarios as it does not require prior knowledge of the workspace geometry. Like much of the other work in this section, this work differs from our own because it assumes the flock mates will be repelled by the shepherd.

9.4.3 Infiltration

The approach presented in this dissertation of using influencing agents that appear to the flock as friendly, homogeneous flock mates is a form of infiltration. In this section, we consider the related work that addresses problems that are most closely related to the problems addressed in this dissertation.

Han *et al.* study how one agent can influence the direction in which an entire flock of agents is moving [42]. Similarly to the work in this dissertation, in their work each flocking agent follows a simple control rule based on its neighbors. They present a simple model that works well in cases where the flocking agents reflexively determine their behaviors in response to a larger team. However, unlike in our work, they only consider one influencing agent with unlimited, non-constant velocity. By having unlimited, non-constant velocity their influencing agent is able to move to any position in the environment within one time step. Unlimited velocity is an unrealistic assumption though, so in our work we assume the agents have bounded velocity.

Jadbabaie *et al.* build on Vicsek *et al.*'s work [80] but use a simpler direction update [50]. They show that a flock with a controllable agent will eventually converge to the controllable agent's heading. Like us, they show that a controllable agent can be used to influence the behavior of the

other agents in a flock. Su *et al.* also present work that is concerned with using a controllable agent to make the flock converge eventually [75]. In Su *et al.*, some flocking agents are specifically informed about which agent is the "controllable agent," whereas in our work we assume the flock will see the influencing agents as normal members of the flock. Our work is different from the work of Jadbabaie *et al.* and Su *et al.* in that while they influence the flock to converge to a target heading eventually, we influence the flock to converge to a target heading quickly by using the *1-Step Lookahead* influencing agent behavior in Algorithm 5.

Celikkanat and Sahin use the same model as Turgut *et al.* [79] and extend it to include informed agents that guide the flock by their preference for a particular direction [18]. In particular, Celikkanat and Sahin study the control of a flock towards a desired direction by the external guidance of informed individuals that have a preferred direction. They conduct various experiments in simulation and on Kobots in which they analyze the controllability of the flock while varying (1) the weight of the direction preference vector of the informed individuals, (2) the ratio of informed individuals in the flock, and (3) the size of the flock. Their flocking model uses a weighted vector sum of three terms: a heading alignment vector (average heading of neighbors), a proximal control vector (maintain cohesion while avoiding collisions), and a preferred direction vector (go in preferred direction, only for informed individuals). They present three metrics for quantifying the steering performance of a flock: (1) mutual information - considers how similar a randomly selected non-informed agents heading is to a randomly selected informed agents heading, (2) accuracy - angular deviation of the direction of the flock from the desired direction, and (3) largest aggregate - combination of the ratio of the largest cluster of agents to the entire flock and the ratio of informed agents in this cluster. As far as we know, Celikkanat and Sahin's work is the only work in this area that mentions the importance of where the informed agents are placed in the flock (which we consider in Chapters 5 and 6) — but this consideration was just an aside in their work. Their work is distinct from our work in that we consider how to control agents from the perspective of knowing how other agents will react, whereas their work simulated flocks and informed agents with predefined behaviors (via the preferred direction vector of the informed agents).

Ioannou *et al.* show that in golden shiner fish, an informed agent must trade off between

trying to get to food as quickly as possible and attempting to guide the rest of the school to the food in order to not leave fish that are not aware of the food source behind [49]. Their real-world experiments on fish show that exhibiting only goal oriented behavior (get food) did increase the accuracy and speed at which uninformed individuals can be led to the food, but that this type of behavior often resulted in the uninformed individuals not being led to the food. These results agree with the results we have seen in our *Maneuver* case experiments in Section 4.5.2, where we lost some of the flock if the influencing agents attempted to influence the flock to turn too quickly. Their work is similar to the work in this dissertation, but unlike the work in this dissertation, their research does not consider how to control some agents and instead utilizes fish that have been trained to find food in specific locations.

Couzin *et al.* considers how groups of animals make informed unanimous decisions [19]. They show in simulation that only a very small proportion of informed agents is required, and that the larger the group the smaller the proportion of informed individuals needed to orient the group. Similar to our work in Section 6.2.3, Couzin *et al.* also considers the trade off between the desire to travel in a particular direction and the desire to influence other agents. They found that when the proportion of informed agents is not small or large, the weighting of the informed agents' preferred direction vector was strongly positively correlated with group cohesiveness and orientation success. Our work is different from the work of Couzin *et al.* in that we consider how to control agents from the perspective of knowing how the other agents will react, whereas they simulate groups with pre-defined behaviors.

Cucker and Huepe propose two Laplacian-based models for balancing the trade off between an informed individual's preference to go in a particular direction and the desire for social interaction [21]. They consider whether the system will converge or break, as well as how effective the system is in heading as a group towards the preferred direction of the informed individuals. The work in this dissertation is different from their work in that we consider how to control agents from the perspective of knowing how other agents will react, whereas their work simulated groups with pre-defined behaviors. Additionally, their work ran many experiments considering various agent velocities, whereas our work assumes that all agents use one constant velocity.

Ferrante *et al.* utilized "information-aware communication" for coordinating movement of

a flock towards a common goal [29]. Specifically, "information-aware communication" means that informed robots communicate the goal direction while uniformed robots communicate the average of messages received from their neighbors. They showed that their "information-aware communication" outperformed the heading communication baseline (in which each agent communicated its own heading) in all simulated cases — including in a case where only one informed agent was used. Their work mainly differs from the work in this dissertation in that we actively consider how to control agents from the perspective of knowing how the other agents will react, whereas in their work each agent communicates in a fixed way based on its type.

In other work, Ferrante *et al.* propose a self-adaptive communication strategy for controlling the heading of a flock using a small set of informed agents [30]. Their work differs from ours in that our informed agents indirectly communicate their actual current headings based on their behaviors, whereas in Ferrante *et al.* the agents do not truthfully report their current headings in some situations in order to game the system. Specifically, they update the degree of confidence a robot has about its possessed information via the use of a local consensus vector. The local consensus vector measures how close the information received by the other robots is to each other and to the information sent by the robot itself. If local consensus was low, it means that there is a conflicting goal direction in the swarm (and hence the robot should have less confidence in its possessed information even if it previously believed it was an informed agent). In their experiments they show that their proposed communication strategy performs better than [29] and [79]. However, unlike the work in this dissertation, Ferrante *et al.* did not consider how to control some agents from the perspective of knowing and planning for how the other agents will react. Instead, the agents behave in a fixed way that is pre-decided or based on type.

Landgraf *et al.* built a wheeled robot that moves a guppy on a pole inside a fish tank [56]. They designed various set behaviors for this robot and showed that after the guppy on a stick has been in the environment for a long time, the robot was able to integrate itself into guppy shoals as well as encourage guppy groups to traverse exposed areas that they would normally avoid. This work is different from the work in this dissertation in that although some of the behaviors involved the robot guppy attempting to "join" the school, it did not join the school with the intention of influencing the behavior of the school through its actions within the school.

9.5 Summary

In this chapter, we described the related work that is most relevant to the work presented in this dissertation. For each related piece of research, we explicitly noted how the work in this dissertation is different or addresses a distinctly different problem.

In Section 9.1 we discussed related work from the broad research area of multiagent coordination and teamwork. Then, in Section 9.2 we considered the most related research on ad hoc teamwork, or teamwork without prior coordination. In Section 9.3 we described research in the flocks, herds, and swarms communities that is most relevant to the work in this dissertation. In particular, we considered work on the two main types of flock formations: cluster formations (Section 9.3.1) and line formations (Section 9.3.2). Finally, in Section 9.4 we described related work on influencing flocks. Within this section, we considered three types of influence: human-led (Section 9.4.1, shepherding (Section 9.4.2), and infiltrating (Section 9.4.3). Section 9.4.3 considered related work on problems that are most similar to this ones addressed in this dissertation — however, none of this work fully addressed the problem considered in this dissertation of utilizing influencing agents to influence a flock towards a particular behavior.

This chapter provided a review of the work most related to the research described throughout this dissertation. Chapter 10 will conclude this dissertation by (1) summarizing the work presented in this dissertation and (2) describing fruitful directions for research that could extend upon the work presented in this dissertation.

10. Conclusions and Future Work

Exciting news emerged from Edmonton International Airport in early May 2017: the airport started a three-month test utilizing a robot bird to encourage native birds to stay away from the airport.[1] The test is using a pilot-operated Clear Flight Solutions Robird[2] during the weekdays and a live falcon during the weekends. The Robird is being tested in conjunction with other bird-deterrents, including baffles to prevent roosting and noise machines to disturb the birds.

The three-month test at Edmonton International Airport is extremely promising as it is the first instance of a robot bird being integrated into daily airport operations. However, a human pilot should not be necessary as algorithms can govern robot behavior and geo-fencing technology can guarantee that the robot bird stays out of active flight paths. Additionally, attempting to scare native birds away from the airport is not optimal because (1) it is impossible to predict where the birds will flock instead and (2) it is stressful to the native birds. The methods presented in this dissertation introduce a more environmentally-friendly way in which flocks could be diverted away from airports. Specifically, this dissertation presents algorithms and methods for using influencing agents to influence flocks towards a particular behavior such as flying around an airport. Although most of this dissertation evaluated these algorithms in simulation, we implemented some of our algorithms on bipedal robots (Chapter 8). Additionally, one of the areas of future work proposed in this chapter is to extend the work in this dissertation for use on robot birds (Section 10.2.6).

In Section 10.1, we review this dissertation's scientific contributions to the areas of multi-agent systems, swarming, and ad hoc teamwork. In Section 10.2 we discuss promising directions for future work that are motivated by the scientific contributions of this dissertation. Finally, in Section 10.3 we provide some concluding remarks.

[1] https://www.utwente.nl/en/news/!/2017/5/121321/robird-to-be-deployed-at-canadian-airport
[2] https://clearflightsolutions.com/methods/robirds

10.1 Contributions

This dissertation presents a complete set of algorithms and methods for using influencing agents to influence a flock towards a particular behavior. Specifically, this dissertation provides the following contributions:

- Our **problem definition** for influencing a flock is presented in Chapter 2. In Chapter 2 we define the assumptions, parameters, and objectives for the problem of adding influencing agents to a flock.

- An **algorithm for leading a stationary flock to a desired orientation** is described in Chapter 3. In Chapter 3 we set bounds on the extent of influence the influencing agents can have on the flock when all of the agents are stationary. We also contribute an algorithm for orienting a stationary flock to a desired orientation using a set of non-stationary influencing agents and analyze this algorithm both theoretically and empirically.

- Directing a flock away from danger requires being able to influence the flock to alter its orientation. As such, three **algorithms for influencing a flock to a desired orientation** are presented in Chapter 4. For each algorithm, we present detailed experimental results. Additionally, we also experimentally consider how to use one of these algorithms to maneuver the flock through turns quickly but with minimal agents being separated from the flock as a result of these turns. Such a behavior could be used to guide a flock around an airport.

- Influencing agents in different parts of a flock have different influence over the flock. Hence, determining how to place influencing agents into a flock if given the opportunity is important. As such, **methods for placing influencing agents into a flock** are considered in Chapter 5. We empirically evaluate each method to determine which placement methods give the influencing agents the most influence over the flock.

- It is not realistic to assume that influencing agents can always be placed directly into a flock. Instead, the influencing agents may need to join the flock from somewhere outside the flock, influence the flock, and then leave the flock. With this need in mind, this dissertation

contributes **methods for influencing agents to join and leave a flock** in Chapter 6. Influencing agents continue to influence their neighbors while joining and leaving though, so these methods must try to decrease the negative influence joining and leaving may have on the flock. We empirically evaluate all of the methods to determine their effectiveness.

- The contributions in Chapters 3-6 assume the influencing agents join a flock that exhibits a particular type of flocking behavior. However, there are many possible variants of flock-member behavior. As such, in Chapter 7 we **evaluate the influencing agent behavior and placement algorithms on flocks with different behaviors**.

- The contributions in Chapters 3-7 are all evaluated in a simulation environment. However, since we believe the work in this dissertation could help reduce bird strikes, we test our **implementation on a robot platform**. Specifically, in Chapter 8 we describe our experiences implementing and evaluating one of the algorithms from Chapter 4 on multiple SoftBank Robotics NAO robots. We report the experimental setup and discuss the experiments in Chapter 8.

10.2 Future Work

Although this dissertation presented effective methods for influencing a flock using influencing agents — including where to place the influencing agents when placement is possible and how the influencing agents can join and leave the flock when placement is not possible — there are still plenty of extensions that could be undertaken. The problem of influencing a group is very rich. In this section, we introduce some of the many extensions that we believe could be fruitful.

We first consider how Chapters 3 through 8 could be extended in Sections 10.2.1 to 10.2.6. Then we consider potential future work related to V-shaped flocks in Section 10.2.7 , to animals and general swarms in Section 10.2.9, and to humans in Section 10.2.10.

10.2.1 Extending Theoretical Contributions

Work in Chapter 3 of this dissertation set bounds on the extent of influence the influencing agents can have on the flock when all of the agents are stationary — but the other chapters of this

dissertation primarily made empirical contributions. In this section, we describe a few possible extensions to theoretical work.

Minimal Bound on Influencing Agents

In Chapter 5 we considered where to place influencing agents in a flock. An interesting theory-based extension to this placement work would be to place a minimal bound on the number of influencing agents required to initially directly influence every flocking agent.

This problem of finding the minimal number of influencing agents to influence all of the flocking agents is equivalent to the set cover problem.[3] Specifically, given a set of flocking agents we can determine (1) whether a particular set of influencing agents' positions and orientations covers all flocking agents and (2) whether there are any influencing agents that could be removed while still maintaining full coverage of the flocking agents. However, determining the optimal coverage is hard, as optimizing the set cover problem is hard. Although finding the optimal coverage is hard, there may be greedy approximation algorithms that could find an approximate answer.

The set cover representation would likely work best for flocking models that use a *visibility sector* neighborhood model — but it would be interesting to consider how the type of neighborhood model used affects the guarantees that can be made.

Performance Guarantee

Another interesting theoretical extension of the work in this dissertation would be to consider when guarantees can be made regarding the flock's performance. In particular, can guarantees can made regarding the flock's convergence to θ^* or the minimum/maximum number of flocking agents that could become lost? When guarantees can be made, can we bound how far these guarantees are from the average case?

As with considering the guarantees on the minimum number of required influencing agents, it would also be interesting to consider how different flocking models affect the guarantees we can make regarding performance guarantees.

[3] Thanks to Noa Agmon for this insight.

10.2.2 Extending Influencing a Flock to a Desired Orientation

Chapter 4 considered the problem of influencing a flock with a *known* flocking model to orient towards a particular orientation. Specifically, in Chapter 4 we considered how to *orient* a flock to a target heading and how to *maneuver* a flock through turns. In this section, we consider how work on influencing a flock towards a desired orientation could be extended when the flocking model is *known*. Later in this chapter, Section 10.2.5 considers extensions when the flocking model is *unknown*.

Additional Comparisons to Other Influencing Agent Behaviors

In Section 4.4.1 we discussed two baseline influencing behaviors. One of these baseline behaviors was modeled after work by Jadbabaie, Lin, and Morse [50] while the other was inspired by Algorithm 2 in Chapter 3. Although these two baselines provided a representative sample of existing methods, there are many other influencing agent behaviors discussed in Chapter 9.

Yu *et al.* 's implicit leadership algorithm [82] will certainly influence a flock to reach a desired orientation slower than the algorithms presented in Chapter 4. Likewise, there are other methods for influencing a flock described in Section 9.4 that would also be good comparison methods. It would be interesting to compare the performance of these methods to those presented in Chapter 4.

Efficient, Deeper Lookahead Searches

In Chapter 4 we presented a 1-Step lookahead algorithm (*1-Step Lookahead*) in Algorithm 5 of Section 4.1 and a 2-step lookahead algorithm (*2-Step Lookahead*) in Algorithm 6 of Section 4.2. We found the complexity of *1-Step Lookahead* to be $O(numAngles * numAgents^2)$ and the complexity of *2-Step Lookahead* to be $O(numAngles^2 * numAgents^3)$. As reported in Section 4.4.3 and discussed in Section 4.4.4, we found that while *2-Step Lookahead* had a much worse complexity than *1-Step Lookahead*, the two methods performed approximately the same. We expected that deeper searches would (1) be computationally infeasible and (2) would perform no better than *1-Step Lookahead*. It was based on this result and analysis that we decided to utilize *1-Step Lookahead* throughout the

remainder of the dissertation to govern behavior of the influencing agents.

However, especially under different environments or in different domains, a deeper lookahead search could be fruitful. Towards this idea, we believe it would be worthwhile to (1) consider how to make the lookahead searches more efficient and (2) implement deeper lookahead searches. The first step towards this type of work would be to design an environment or domain in which deeper searches would be useful. The second step would be to consider how the algorithms can be made more efficient, perhaps through approximation.

Influence to Not Lose Neighbors

All of the influencing agent algorithms in Chapter 4 considered how to influence the flocking agents towards θ^* quickly. However, it could be interesting to instead consider how to influence the flocking agents towards θ^* while not losing influence over any flocking agents that are not already oriented towards θ^*.

One initial idea for managing the trade-off between orienting towards θ^* and maintaining influence over flocking agents that still need to be influenced might be to:

- Influence maximally towards θ^* if doing so would not lose any flocking agents that are not oriented towards θ^*
- Otherwise, influence the neighbors towards θ^* as much as possible without losing influence over any influencing agents that are not oriented towards θ^*

Avoid Obstacles Automatically

Section 4.5 described using the *1-Step Lookahead* behavior to *maneuver* a flock around an obstacle. In our experiments, this was done by maneuvering through a set of turns along a flexible but predefined path. However, it would be better if the influencing agents could determine autonomously the approximate path and when to influence the flock to turn.

At a high level, the flock would need to detect or know based on prior knowledge where obstacles exist in the environment. The location of stationary obstacles could be provided to the robots, whereas dynamic obstacles would need to be detected. Once the obstacles are detected/known, the

influencing agents would need to determine how long it will take to turn the flock. The time needed to turn the flock would depend on the size and density of the flock, the flocking model utilized by the flock, the number of influencing agents available, and potentially other factors. Once the flock's turning ability is known, then the influencing agents must determine the path to take around the obstacle(s). If the obstacles are all known ahead of time, this path could be optimized — but if obstacles are detected locally in real-time, then the path around each obstacle must be determined as the obstacles are detected.

Other Coordinated Behaviors

Although Section 4.3 presented a *Coordinated* influencing agent behavior, other coordinated behaviors may be superior. The behavior presented in Algorithm 7 of Section 4.3 pairs influencing agents with shared neighbors and determines the behavior of both influencing agents together, such that their joint influence on shared neighbors is considered.

Surprisingly, results in Section 4.4.3 showed that the *Coordinated* behavior did not perform significantly better than the *1-Step Lookahead* behavior. As we discuss in Section 4.4.4, performance was likely similar since the behavior of influencing agents under the *Coordinated* behavior is often similar to under the *1-Step Lookahead* behavior. The observed similarity in performance between the *Coordinated* behavior and the *1-Step Lookahead* behavior could mean that the *1-Step Lookahead* behavior really is best — but it could also mean that coordinated behavior can be improved.

There are many potential ways to coordinate influencing agent behavior. One place to start would be to globally coordinate behavior of all of the influencing agents. Determining how well global coordination can perform would set an upper bound on how well coordinated behaviors can perform.

Influencing Multiple Flocks

Although the work in this dissertation only considered influencing a single flock, it is possible that multiple flocks may need to be influenced within the same space. Although these flocks could be influenced separately, there may be benefits to coordinating the influence. Likewise, it could be beneficial to influence multiple flocks to join. It would be interesting to consider when joining would

be beneficial, as well as the required behavior by the influencing agents to facilitate this joining.

Flocks may also naturally cross paths while flocking. These flocks may join naturally as their flight paths intersect. Even if the flocks do not join naturally, it would likely be easy to encourage the flocks to meld into one flock when their flight paths cross. In the case where two flocks naturally cross flight paths, it could be interesting to study (1) the influencing behavior required to meld two flocks into one flock as well as (2) the influencing behavior required to keep the two flocks separate.

10.2.3 Extending Placing Influencing Agents into a Flock

Chapter 5 considered various methods for placing influencing agents into a flock. Although these methods —especially the cluster placement methods — performed well, it is possible that other methods could perform better or be more computationally efficient. In this section, we describe multiple potentially fruitful extensions to our influencing agent placement work.

Scaling up the Graph Placement Method

The *Graph* placement method described in Section 5.3 was one of the best performing placement methods considered in Chapter 5. Unfortunately, the $O(n^3\binom{m^2+m}{k})$ computational complexity meant that we were unable to run it for large tests on flocks with more than $k = 10$ influencing agents. Remember from Section 2.1.2 that m refers to the number of flocking agents, k refers to the number of influencing agents, and n refers to the total flock size (i.e. $n = m + k$).

Due to the computational complexity of the *Graph* placement method discussed in Section 5.3, a significant extension to this dissertation would be to scale up the *Graph* placement method such that it can be quickly executed by flocks with both more influencing agents and more flocking agents. One initial idea for scaling up the *Graph* placement method is to prune the number of possible influencing agent positions. Specifically, positions that are (1) within the neighborhood of other positions, (2) not influential enough (i.e., only have limited neighbors), or (3) have neighbors with an average current orientation near θ^* could be removed. Reducing the number of possible influencing agent positions will significantly reduce the number of sets of k influencing agents that must be considered.

Improving Hybrid Placement Methods

The *hybrid* placement methods described in Section 5.4 utilized the *Graph* placement method to place the first k_g influencing agents and then used a more computationally efficient method to place the remaining $k - k_g$ influencing agents.

It would be interesting, and potentially fruitful, to consider whether there are situations in which it would be better to place $k - k_g$ influencing agents using a constant-time placement method first. Specifically, the constant time method could select $k - k_g$ well-spaced positions and then the *Graph* method could select k_g placements that cover the most critical areas not already covered by the influencing agents placed using the constant-time placement method.

Placement Selection based on Agent Heading

All of the influencing agent placement methods in Chapter 5 were based on placing influencing agents in areas where they could influence flocking agents. However, none of the methods in Chapter 5 considered flocking agent *heading* when determining placement.

Considering the heading of flocking agents while determining where to place influencing agents could significantly improve performance. Intuitively, we would strongly prefer to only place influencing agents within the neighborhoods of flocking agents that *need* to be influenced. An initial step could be to not place influencing agents within the neighborhoods of flocking agents that are already oriented close to θ^*. One potential issue is that the flocking agents initially oriented close to θ^* could be pulled away from θ^* by other flocking agents — but being pulled away might not be problematic as these flocking agents would hopefully be influenced by influencing agents.

Placement Methods for Different Neighborhood Models

Although many of the placement methods described in Chapter 5 would perform well across a variety of neighborhood models, particular placement methods inherently perform better for particular neighborhood models. As such, it would be interesting to design particular placement methods that perform especially well with particular neighborhood models. One initial idea would be to try placing influencing agents (1) on the borders or (2) in front of the flock when the flocking agents

use a *visibility sector* neighborhood model.

Learning Placements

None of the algorithms or methods in this dissertation utilize learning — doing so is certainly a potentially fruitful extension. Although we discuss potential extensions towards learning the correct flocking model in Section 10.2.5, in this section consider the idea of learning effective influencing agents placements when interacting with the same species (for which we have an accurate model) repeatedly. Repeated interaction could first allow the influencing agents to determine what type of placement method performs best in particular scenarios. Given time for more interactions, the influencing agents could then optimize positioning parameters within the best performing placement method.

Wait for Convergence before Determining Placements

The experiments in Chapter 5 placed influencing agents into a flock in which all of the flocking agents had random headings within 90 degrees of θ^*. However, it is possible that performance would be improved if we waited for the flock to converge to a heading before determining influencing agent placements.

In particular, we expect that the flock would converge into one or more flocks, where each flock would be flocking towards a particular orientation. Once these flocks are stable, at least one influencing agent could be added into each flock. The influencing agent(s) could then influence the flock to change course and flock towards θ^*.

Local Position Selection

All of the placement methods described in Chapter 5 assume that the influencing agents have perfect global information regarding the placement of all of the flocking agents and that the placements of the influencing agents can be determined centrally. Although neither of these assumptions are unrealistic considering the availability of avian radar and wireless networks, there will likely be situations in which only local sensing is available. As such, it is important to consider placement methods that do not assume perfect global information.

If global information via avian radar is not available, then the influencing agents would need to use other sensors and communicated information to create a local understanding of where flocking agents are currently located. Assuming the influencing agents are able to pre-coordinate, set regions of the flock could be allocated to each influencing agent. The influencing agents would then have to determine where to place themselves within their allocated region using local information.

10.2.4 Extending Joining and Leaving a Flock

The approaches presented in Chapter 6 for joining and leaving a flock considered how to join and leave a flock while minimizing the negative influence on the flock during joining and leaving. Although the approaches in Chapter 6 performed reasonably well in experiments, there may be other approaches that perform better. Hence, in this section we consider a few potentially fruitful extensions.

Picking a Path by which to Enter a Flock

All of the *intercept* approaches for joining a flock presented in Section 6.1.2 call for the influencing agents to join exterior areas of the flock. The *intercept* approaches had the influencing agents join exterior areas of the flock because we found that travelling to positions inside the flock usually caused the flock to disperse before the influencing agents reached their desired positions. However, it would be beneficial to find a way for influencing agents to reach positions inside the flock.

Influencing agents could alternate behaviors while joining, similarly to the *influence while leaving* approach from Section 6.2.3. This could be accomplished by having the influencing agents alternate between mitigating their behavior and trying to enter along a particular path. The following metrics could be considered while determining the path for each influencing agent: (1) influences the fewest flocking agents, (2) influences the fewest flocking agents incorrectly / away from θ^*, (3) minimizes dispersion of the flock. Along the path, the influencing agents could alternate between influencing, moving along the path, and condensing the flock.

Leaving in Pairs

In Section 6.2 we presented three approaches for influencing agents to utilize when attempting to leave a flock while minimizing the negative effect on the flock of leaving. Although all three approaches worked well in particular situations, there may be other leaving approaches that would work better in some situations.

In particular, it could be fruitful to determine whether there are situations where having influencing agents leave in pairs could be beneficial. For example, consider a dense flock with influencing agents positioned along the flock's borders. In such a situation, the influencing agents may be able to minimize their negative influence on the flock by leaving in opposite directions at the same time.

Morphing Agents

Finally, a slightly different situation could be investigated: one in which the influencing agents are able to go from being recognized as self to other. If the influencing agents are able to morph into predators, they could potentially "leave" the flock easily by scaring the flock away while at a border of the flock. Likewise, if the influencing agents were able to morph between sensed and undetected, then the placement methods from Chapter 5 could be used with the influencing agents only being recognizable while influencing. Although morphing may seem infeasible, it is possible that robot birds could be designed to morph between friend and foe. As mentioned in Section 6.1.2, the makers of the Clear Flight Solutions Robird[4] stated at the 2015 North American Bird Strike Conference that birds believe another bird is "one of its own" if its wing movement and silhouette are the same. Hence, a robot bird could potentially morph between friend and foe by simply changing its silhouette and wing movement.

10.2.5 Generalizing to Different Flocking Models

Chapter 7 evaluated how well the algorithms and methods in this dissertation performed when different flocking models were utilized. In this section we consider how the work in Chapter 7 could be extended.

[4] `http://clearflightsolutions.com/methods/robirds`

Alternate Neighborhood Models

In Section 7.2.4 we considered how well the methods and algorithms in this dissertation performed using four different neighborhood models. However, there are still many other neighborhood models that could be considered.

In particular, it would be interesting to consider the following neighborhood models:

- A weighted influence neighborhood model in which the orientation, speed, and position of farther agents is less accurate.
- The zonal neighborhood model used in Tiwari *et al.* assumes an agent a_i is influenced by neighbors within three spherical zones [78]. Within the zone nearest a_i, a_i and its neighbors are repelled by each other. Within the second zone, a_i and its neighbors orient in the same direction of motion. Within the zone farthest from a_i, a_i and its neighbors attract each other.
- A neighborhood model that represents the idea by Rosenthal *et al.* that connectivity is weighted, directed, and heterogeneous [70]. Rosenthal *et al.* showed that individuals with relatively few strongly connected neighbors are both most socially influential and most susceptible to social influence.

For each of these neighborhood models, as well as any additional models, it would be interesting to (1) consider how the algorithms and methods in this dissertation perform when using these models and (2) as needed, extend the algorithms and methods in this dissertation to perform well with these models.

Handling Flocking Model Noise and Instability

In this dissertation, we assume that we know the flocking model being utilized by the flocking agents and that there is no noise or instability in this flocking model nor the environment. Both of these assumptions are substantial, so it would be interesting to consider how well the algorithms and methods described in this dissertation handle noise and instability — as well as how these algorithms and methods can be extended to better handle noise and instability.

Adding noise to the influencing agents' estimation of the flocking agents' locations and orientations would be an ideal starting point as this type of noise should merely decrease performance based on the amount of noise. Noise in orientation and position for all agents during flocking could potentially be caused by wind currents. This type of noise could require a flock that was previously converged to θ^* to need to be influenced again if they stray too far from θ^*.

There could also be noise in the detection of surrounding agents. As such, it would be interesting to consider how much temporarily undetected agents affect the behavior of both the flocking agents and the influencing agents.

Goal-oriented Flocks

Throughout this dissertation we assumed that the flocking agents would use a set flocking model, but that they had no set goal direction — instead their heading was a factor of the current orientations of their neighbors. Although assuming that flocks have no set goal direction may not seem realistic, we have found no work that attributes any external goal behavior or direction to a flock. That being said, many flocking models do consider how a flock will act in the presence of a predator — so one way to attribute a goal to the flock would be to consider its behavior when being chased by a predator.

It would be particularly interesting to (1) analyze how well the methods in this dissertation work when a flock is being chased by a predator and (2) consider how these methods can be extended to perform better both while a flock is being chased by a predator as well as after the chase has ended.

Inferring and Generalizing Flocking Models

Although we assume throughout this dissertation that the flocking model of the flocking agents is known by the influencing agents, this assumption may not always be reasonable. As such, it would be useful to consider (1) how to infer what flocking model a flock is employing and (2) how best to influence a flock utilizing this model. If the influencing agents know how to influence flocking agents utilizing a variety of flocking models, then a simple solution is to classify the flock's behavior to one of the known models and behave accordingly.

Performance would be better, though, if the influencing agents were able to generalize their behavior to new, unknown flocking models. One such method of doing so is described in the next section.

Learning Flocking Models and How to Influence

One potentially fruitful extension to the work presented in this dissertation is to approach the problem of influencing a flock from a machine learning perspective. This type of extension could potentially be the basis for another entire dissertation. Most flocking models are parametrizable, which makes them well suited for learning.

The influencing agents could be designed to first learn the flock's flocking model and then learn how to best influence the flock based upon this flocking model. If the flocking agents are given the ability to exhibit various real-life flocking models, the influencing agents could potentially even learn in simulation how to best influence a variety of real-life species. Additionally, if the flocking agents can behave according to real-life flocking models, then various assumptions — such as whether a particular species uses a 7-nearest neighbors neighborhood model or a distance-based neighborhood model — could be tested in simulation by comparing simulated behavior to behavior observed in nature. In this way, learned flocking models could be used to validate flocking models proposed by biologists. We discuss extensions towards validating theories of biologists in Section 10.2.9.

10.2.6 Extending Robot Implementation

This dissertation focused on using influencing agents to influence flocks towards particular behaviors. The research in this dissertation could be extended to autonomously influence birds away from airports using robot birds. In this section, we discuss some of the extensions that would first need to be undertaken.

Robot Birds

As mentioned in Chapter 1, resident birds in the Netherlands attempted to flock with early prototypes of Clear Flight Solutions' Robird. However, as far as we know, only one project has captured

video[5] of natural birds flocking with a robot bird: the Zapdos project by Hithesh Vurjana, Amritam Das, and Apurva Bhattad of SRM University in Chennai, India. Unfortunately, this project seems to have concluded in 2014 with no publications resulting. Although the robot bird was remote controlled in the video, the video shows that flocks of birds will flock with a robot bird if the circumstances are right.

The designers of the Clear Flight Solutions Robird stated at the 2015 North American Bird Strike Conference that flocks will react to a robot bird as one of their own if the robot bird has a silhouette and wing flap motion similar to their own. However, it would be critical to validate that the species of bird that needs to be influenced will flock with the particular robot bird(s) that will be utilized. The design may need to be modified multiple times before acceptance by the native birds is obtained.

The Clear Flight Solutions Robird — which has minimal sensors on-board — currently has a battery life of just 12 minutes. Robot birds that would be used to influence flocks would need additional sensors and communications hardware, so the maximum flight time would likely be less. Hence keeping weight down and maximizing battery capacity will likely be critical aspects in designing robot birds that can influence flocks.

Supporting Technology

The algorithms in this dissertation assume (1) perfect global location and orientation information, (2) knowledge of the flocking agents' flocking model and (3) communication between influencing agents.

Global positioning system data could provide location and orientation information for all influencing agents. Meanwhile, avian radar could be used to get high-accuracy location and orientation information for all natural birds. However, as of March 2015, avian radar was only installed at two commercial airports in the United States: DFW in Dallas, TX and SEA in Seattle, WA. Most airports do not have avian radar, so it would be important to extend the algorithms in this dissertation to handle imperfect information. The algorithms could be further updated to handle local information, although it would likely be better to use communicated information from all of

[5] https://www.youtube.com/watch?v=iTsi9F7kEjE

the influencing agents to create a shared world model.

Throughout this dissertation we assume that we know the flocking model being utilized by the flocking agents. If we know the type of species the influencing agents need to influence ahead of time, this assumption is not problematic. However, this requires that either the influencing agents, other systems, or humans determine what type of species the influencing agents need to influence. As such, it would be helpful for the influencing agents to be able to detect and recognize specific species automatically.

Policy

Finally, once the robot and software technologies are ready for deployment, it would be necessary to find an airport or air force base that is willing to allow tests to be conducted. At the very least, this airport or air force base would need assurances that the system is safe and will not negatively affect their daily operations. It would be best to monitor the number of bird-related incidents for a significant period before influencing agents are introduced to establish a baseline. Once the baseline is established, it would be best to alternate days or weeks in which the influencing agents are utilized so that seasonal differences in bird activity do not inaccurately affect the results.

10.2.7 Extensions to Line Formation Flocking

As explained in Section 9.3, there are two main types of flocking formations: cluster formations and line formations. This dissertation considered how to influence flocks behaving according to Reynolds' flocking algorithm, which assumes cluster formations. In this section we consider how influencing agents could influence V-shaped flocks and describe some future directions that we believe could be fruitful.

Understanding Line Formation Flocking

Section 9.3.2 discusses multiple line flocking models — based on our understanding of biologists' current knowledge of line formations, the V-flocking model by Nathan and Barbosa seems to be biologically realistic [65].

The V-flocking formation is a direct result of aerodynamics. Specifically, as a bird flaps its wings, a rotating vortex of air rolls off each of its wingtips. These vortices constantly push the air immediately behind the bird downward (downwash) and the air off to the sides upwards (upwash). When another bird is able to fly with its wingtip in either of the upwash zones, it gets free lift and hence does not need to expend as much energy. Alternatively, if its wingtip is in the downwash zone, it must extend extra energy to avoid being pushed down. In order to save energy, every bird attempts to place its wingtip in an upwash zone created by the bird ahead of it.

V-flocks are usually made up of a family group, or in the case of large flocks, multiple family groups. Within these flocks, strong, experienced family members take turns leading the flock at the tip of the V-formation. When the bird at the tip of the V-formation becomes tired, it will rotate back into the formation and another capable family member will take the helm. Interestingly, if a bird in a V-flock becomes sick or injured, another bird from the flock will always stay behind with it. These birds — or just one bird if the sick or injured bird dies — will join other flocks while catching up with their own family's flock.

10.2.8 Sweet Spots for Influence

Unlike cluster formations, the heading of a V-flock is entirely controlled by the bird at the tip of the V-formation. Since only experienced family members take turns leading the flock, non-family members will not be given the opportunity to advance to the lead position. Hence, it will be difficult to influence a V-flock by becoming lead bird — but there may be other ways in which an influencing agent could join and influence a V-flock.

Nathan and Barbosa [65] list three basic rules for V-flocking. Within these rules, there may be sweet spots for influencing a subset of the flock. For example, it might be possible to gain control of the flock behind an influencing agent if another influencing agent blocks the view of the updraft bird. Another example would be to have an influencing agent slowly diverge from the flock. This influencing agent might be able to take the flocking agents that are behind it as it diverges, and then act as leader of the new sub-flock. It is unclear whether either of these strategies would work in simulation or in real life though.

10.2.9 Extensions to Other Animal Domains

The algorithms and methods described in this dissertation were designed for influencing agents to influence a flock of birds towards a particular behavior. However, there are common elements in many types of group behavior. As such, in this section we consider future directions that could apply the work in this dissertation towards other types of animals.

Influencing Endangered Species towards Safety

Reynolds' model represents general flocking, herding, and schooling behaviors. Hence, many of the methods and algorithms from the dissertation could apply to other species with minimal changes given the behavior models of a new species. One particular application that could be interesting would be to consider which types of endanger species can be influenced using influencing agents.

There are many endangered animals that live in national parks and other protected areas, but occasionally roam outside of the protected boundaries. Yellowstone National Park's gray wolves are protected on park land, but many of the states surrounding Yellowstone allow hunting of the gray wolves once they leave park land. Wolves are pack animals, but it is unclear whether a robot wolf could integrate into a pack and influence the behavior of the pack as a whole. Nonetheless, considering whether species of interest could be influenced by robot influencing agents integrating into their pack/herd/family would certainly be an interesting and potentially high impact extension to the work in this dissertation.

Validating Theories of Biologists

When reading papers on how a particular species flocks, it quickly becomes apparent that oftentimes the biologists do not agree — one example is disagreement regarding neighborhood models as described by Ballerini *et al.* and Bialeka *et al.* [6, 13]. The work in this dissertation could be extended to validate some of the theories from the biologists. Specifically, if biologists say that the seven nearest agents are seen as neighbors by a specific species, this could be tested in simulation. If the simulation dynamics and behavior matches flocking behavior observed in nature, then the theory of seven nearest neighbors would be validated.

Additionally, if there are multiple flocking models that biologists claim are most realistic for a specific species, simulation tests could be run to determine which flocking model most closely matches real-life behavior for the species.

Managing a Destructive Swarm

Section 9.3 described work by Barca to create an anti-swarming strategy that could be utilized against adversary robot swarms [7]. The work in this dissertation could be extended to influence dangerous swarms into a safe area or to a safer behavior, assuming that the robot swarm saw our influencing agents as other swarm members.

Behaviors such as were shown in the *maneuver* experiments in Section 4.5 could be used to guide a destructive swarm around a sensitive area or perhaps into a safe area where they could be disabled. Additionally, the influencing agents could potentially disperse the destructive swarm. Throughout most of this dissertation we focused on keeping the flock together, but as we saw in the *leaving* experiments in Section 6.4 it should be relatively easy to purposefully disperse the flock. One simple method would be to merely have the influencing agents simultaneously "leave" the flock in opposite directions.

10.2.10 Extensions to Human Domains

In 2012 Grosz wrote an article in AI Magazine entitled "What question would Turing pose today?" [39]. In this article, her answer to the question posed in the title was

> Is it imaginable that a computer (agent) team member could behave, over the long term and in uncertain, dynamic environments, in such a way that people on the team will not notice it is not human.

In many ways, being able to "pass" this test would be creating an influencing agent to influence the human team without the human team realizing the agent was anything other than an average teammate.

However, there are many other ways in which the algorithms and methods in this dissertation could be used to influence humans towards a desired behavior. In this section, we consider a few

possible extensions.

Pedestrian Behavior

Bonneaud and Warren present a model for realistic pedestrian behavior [14]. Their model is validated by comparing real-life pedestrian behavior against the behavior observed using their model.

Bonneaud and Warren's model could be used to design behaviors for trained humans or realistic robots such that these humans or robots could influence pedestrians to behave in a particular manner. If an agent passes Grosz's proposed Turing test, then that agent could influence humans just as well (if not better) than a trained human. In particular, it would be interesting to first show that a few trained humans can influence pedestrians to behave in a particular manner consistently. Then the task would be to design a robot agent — both its physical form and its behavior — that encourages the pedestrians to behave in a manner similar to how they behaved when the trained humans influenced them.

Crowd Behavior

When humans are in groups, panic spreads quickly and information is often falsely construed. Evacuating a building during a fire, a plane after a crash, or a festival after gunshots are heard can be chaotic and dangerous.

It is difficult to reenact true crowd panic — the closest evacuation testing is likely the 90-second aircraft evacuation tests that must be passed for each new plane type. In these tests, the aircraft manufacturer must show that a plane filled with crew and passengers can be evacuated in less then 90 seconds. Even in these tests — where nothing is actually on fire and there is minimal incentive for the passengers to exit quickly — up to 5% of the passengers often sustain injuries. As can be imagined, the injury rate during a real crash with fire, smoke, fuel leaks, and darkness would be significantly higher because the level of panic would be significantly higher.

With this in mind, extensions of the work in this dissertation could consider how influencing agents could influence human crowds to act calmly during an evacuation — or perhaps even influence evacuees to take an alternate route to safety. Likewise, the placement methods discussed in Chapter 5 of this dissertation could potentially be extended to determine where to place robots

with sensors and loudspeakers throughout crowds that have the potential to stampede. All of these approaches could be tested in a realistic crowd simulation, or perhaps even in an aircraft or building evacuation test setting.

10.3 Concluding Remarks

This dissertation describes how influencing agents can be utilized in order to influence a flock with a known flocking model toward a particular behavior. In particular, this dissertation makes novel contributions to the field of artificial intelligence — especially the multiagent systems and ad hoc teamwork communities — by contributing influencing agent behaviors for efficiently influencing a flock, placement methods for placing influencing agents into a flock, and methods for joining and leaving a flock in motion without negatively influencing the flock.

By considering how the influencing agents should behave and where the influencing agents should be placed — as well as how the influencing agents should join and leave a flock if placement is not possible — we show that groups can be effectively influenced from within. We hope that the influencing agent algorithms and methods presented in this dissertation will be extended to influence flocks of birds, or other animal species, towards safer behavior than they would naturally achieve. Throughout this dissertation, we have considered the motivating example of using influencing agents to influence flocks of birds away from airports. We hope that the algorithms and methods described in this dissertation will help this motivating example become a reality.

Bibliography

[1] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, page 271. ACM, 2014.

[2] S. V. Albrecht. *Utilising Policy Types for Effective Ad Hoc Coordination in Multiagent Systems*. PhD thesis, The University of Edinburgh, Edinburgh, UK, November 2015.

[3] M. Andersson and J. Wallander. Kin selection and reciprocity in flight formation? *Behavioral Ecology*, 15(1):158–162, 2004.

[4] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[5] I. L. Bajec and F. H. Heppner. Organized flight in birds. *Animal Behaviour*, 78(4):777–789, 2009.

[6] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic. Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proceedings of the National Academy of Sciences*, 105(4):1232–1237, 2008.

[7] J. C. Barca. Anti-swarming from the swarm robotics perspective. October 2016.

[8] S. Barrett. *Making Friends on the Fly: Advances in Ad Hoc Teamwork*. PhD thesis, The University of Texas at Austin, Austin, Texas, USA, December 2014.

[9] S. Barrett, K. Genter, M. Hausknecht, T. Hester, P. Khandelwal, J. Lee, M. Quinlan, A. Tian, P. Stone, and M. Sridharan. Austin Villa 2010 Standard Platform team report. Technical Re-

port UT-AI-TR-11-01, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, January 2011.

[10] S. Barrett, K. Genter, Y. He, T. Hester, P. Khandelwal, J. Menashe, and P. Stone. UT Austin Villa 2012: Standard Platform League world champions. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 36–47. Springer, 2013.

[11] S. Barrett, K. Genter, T. Hester, P. Khandelwal, M. Quinlan, P. Stone, and M. Sridharan. Austin Villa 2011: Sharing is caring: Better awareness through information sharing. Technical Report UT-AI-TR-12-01, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, January 2012.

[12] M. Berger, L. M. Seversky, and D. S. Brown. Classifying swarm behavior via compressive subspace learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5328–5335. IEEE, 2016.

[13] W. Bialeka, A. Cavagna, I. Giardinab, T. Morad, E. Silvestrib, M. Vialeb, and A. Walczak. Statistical mechanics for natural flocks of birds. *Proceedings of the National Academy of Sciences*, 109(11):4786–4791, 2012.

[14] S. Bonneaud and W. H. Warren. A behavioral dynamics approach to modeling realistic pedestrian behavior. In *Proceedings of the 6th International Conference on Pedestrian and Evacuation Dynamics*, 2012.

[15] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *Proceedings of the Twentieth AAAI Conference on Artificial Intelligence (AAAI'05)*, pages 53–58. AAAI Press, 2005.

[16] H. Bronnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[17] D. S. Brown, S. C. Kerman, and M. A. Goodrich. Human-swarm interactions based on managing attractors. In *Proceedings of the 2014 ACM/IEEE International Conference on Human-Robot Interaction*, pages 90–97. ACM, 2014.

[18] H. Celikkanat and E. Sahin. Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865, September 2010.

[19] I. D. Couzin, J. Krause, N. R. Franks, and S. A. Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516, 2005.

[20] E. Cristiani and B. Piccoli. A unifying model for the structure of animal groups on the move. 2009.

[21] F. Cucker and C. Huepe. Flocking with informed agents. *Mathematics in Action*, 1(1):1–25, 2008.

[22] R. D'Andrea and P. Wurman. Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities. In *Technologies for Practical Robot Applications, 2008. TePRA 2008. IEEE International Conference on*, pages 80–83. IEEE, 2008.

[23] S. Dasgupta. Performance guarantees for hierarchical clustering. In *15th Annual Conference on Computational Learning Theory*, pages 351–363. Springer, 2002.

[24] K. S. Decker and V. R. Lesser. Readings in agents: Designing a family of coordination algorithms. pages 450–457. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[25] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[26] B. Dias. *Traderbots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.

[27] J. Enright and P. R. Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Automated action planning for autonomous mobile robots*, pages 33–38, 2011.

[28] A. Feldman, M. Hybinette, and T. Balch. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *Journal of Field Robotics*, 29(2):258–276, 2012.

[29] E. Ferrante, A. E. Turgut, N. Mathews, M. Birattari, and M. Dorigo. Flocking in stationary and non-stationary environments: A novel communication strategy for heading alignment. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN'10)*, pages 331–340. Springer-Verlag, 2010.

[30] E. Ferrante, A. E. Turgut, A. Stranieri, C. Pinciroli, M. Birattari, and M. Dorigo. A self-adaptive communication strategy for flocking in stationary and non-stationary environments. *Natural Computing*, 13(2):225–245, 2014.

[31] K. Genter, N. Agmon, and P. Stone. Ad hoc teamwork for leading a flock. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'13)*, pages 531–538. International Foundation for Autonomous Agents and Multiagent Systems, May 2013.

[32] K. Genter, T. Laue, and P. Stone. Three years of the RoboCup Standard Platform League Drop-in Player Competition: Creating and maintaining a large scale ad hoc teamwork robotics competition. *Autonomous Agents and Multi-Agent Systems (JAAMAS)*, pages 1–31, 2016.

[33] K. Genter, P. MacAlpine, J. Menashe, J. Hannah, E. Liebman, S. Narvekar, R. Zhang, and P. Stone. UT Austin Villa: Project-driven research in ai and robotics. *IEEE Intelligent Systems*, 31(2):94–101, 2016.

[34] K. Genter and P. Stone. Influencing a flock via ad hoc teamwork. In *Proceedings of the Ninth International Conference on Swarm Intelligence (ANTS'14)*, pages 110–121. Springer, September 2014.

[35] K. Genter and P. Stone. Ad hoc teamwork behaviors for influencing a flock. *Acta Polytechnica*, 2016.

[36] K. Genter and P. Stone. Adding influencing agents to a flock. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'16)*, pages 615–623. International Foundation for Autonomous Agents and Multiagent Systems, May 2016.

[37] K. Genter and P. Stone. Agent behaviors for joining and leaving a flock (Extended abstract). In *Proceedings of the 2017 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'17)*. International Foundation for Autonomous Agents and Multiagent Systems, May 2017.

[38] K. Genter, S. Zhang, and P. Stone. Determining placements of influencing agents in a flock. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*, pages 247–255. International Foundation for Autonomous Agents and Multiagent Systems, May 2015.

[39] B. Grosz. What question would Turing pose today? *AI Magazine*, 33(4):73, 2012.

[40] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[41] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[42] J. Han, M. Li, and L. Guo. Soft control on collective behavior of a group of autonomous agents by a shill agent. *Journal of Systems Science and Complexity*, 19(1):54–62, 2006.

[43] J. F. Harrison, C. Vo, and J.-M. Lien. Scalable and robust shepherding via deformable shapes. In *International Conference on Motion in Games*, pages 218–229. Springer, 2010.

[44] H. O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2):174–194, 1958.

[45] C. K. Hemelrijk and H. Hildenbrandt. Some causes of the variable shape of flocks of birds. *PLoS ONE*, 6(8), 2011.

[46] J. E. Herbert-Read. Understanding how animal groups achieve coordinated movement. *Journal of Experimental Biology*, 219(19):2971–2983, 2016.

[47] J. E. Herbert-Read, A. Perna, R. P. Mann, T. M. Schaerf, D. J. T. Sumpter, and A. J. W. Ward. Inferring the rules of interaction of shoaling fish. *Proceedings of the National Academy of Sciences*, 108(46):18726–18731, 2011.

[48] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[49] C. C. Ioannou, M. Singh, and I. D. Couzin. Potential leaders trade off goal-oriented and socially oriented behavior in mobile animal groups. *The American Naturalist*, 186(2):284–293, 2015.

[50] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[51] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings 2006 IEEE International Conference on Robotics and Automation (ICRA'06)*, pages 570–575. IEEE, 2006.

[52] S.-Y. Jung, D. S. Brown, and M. A. Goodrich. Shaping couzin-like torus swarms through coordinated mediation. In *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1834–1839. IEEE, 2013.

[53] S. Kerman, D. Brown, and M. A. Goodrich. Supporting human interaction with robust robot swarms. In *2012 5th International Symposium on Resilient Control Systems (ISRCS)*, pages 197–202. IEEE, 2012.

[54] A. J. King, A. M. Wilson, S. D. Wilshin, J. Lowe, H. Haddadi, S. Hailes, and A. J. Morton. Selfish-herd behaviour of sheep under threat. *Current Biology*, 22(14):R561–R562, 2012.

[55] M. Klotsman and A. Tal. Animation of flocks flying in line formations. *Artificial life*, 18(1):91–105, 2012.

[56] T. Landgraf, H. Nguyen, J. Schroer, A. Szengel, R. Clement, D. Bierbach, and J. Krause. Blending in with the shoal: Robotic fish swarms for investigating strategies of group formation in guppies. In *Conference on Biomimetic and Biohybrid Systems*, pages 178–189. 2014.

[57] S. Liemhetcharat. *Representation, Planning, and Learning of Dynamic Ad Hoc Robot Teams*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, August 2013.

[58] J.-M. Lien, O. B. Bayazit, R. T. Sowell, S. Rodriguez, and N. M. Amato. Shepherding behaviors. In *2004 IEEE International Conference on Robotics and Automation (ICRA'04)*, pages 4159–4164. IEEE, April 2004.

[59] J.-M. Lien, S. Rodriguez, J.-P. Malric, and N. M. Amato. Shepherding behaviors with multiple shepherds. In *2005 IEEE International Conference on Robotics and Automation (ICRA'05)*, pages 3402–3407. IEEE, April 2005.

[60] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A multi-agent simulation environment. *Simulation: Transactions of the Society for Modeling and Simulation International*, 81(7):517–527, 2005.

[61] P. MacAlpine, K. Genter, S. Barrett, and P. Stone. The RoboCup 2013 Drop-in Player Challenges: Experiments in ad hoc teamwork. In *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*, pages 382–387, Chicago, IL, USA, September 2014. IEEE.

[62] J. Martin, J. Belant, T. DeVault, B. Blackwell, L. B. Jr., S. Riffell, and G. Wang. Wildlife risk to aviation: A multi-scale issue requires a multi-scale solution. *Human-Wildlife Interactions*, 5(2):198–203, 2011.

[63] J. Menashe, S. Barrett, K. Genter, and P. Stone. UT Austin Villa 2013: Advances in vision, kinematics, and strategy. In *The Eighth Workshop on Humanoid Soccer Robots at Humanoids 2013*, 2013.

[64] C. Moeslinger, T. Schmickl, and K. Crailsheim. Emergent flocking with low-end swarm robots. In *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS'10)*, pages 424–431. Springer, 2010.

[65] A. Nathan and V. C. Barbosa. V-like formations in flocks of artificial birds. *Artificial life*, 14(2):179–188, 2008.

[66] A. Pierson and M. Schwager. Bio-inspired non-cooperative multi-robot herding. In *2015 IEEE*

International Conference on Robotics and Automation (ICRA'15), pages 1843–1849. IEEE, May 2015.

[67] S. J. Portugal, T. Y. Hubel, J. Fritz, S. Heese, D. Trobe, B. Voelkl, S. Hailes, A. M. Wilson, and J. R. Usherwood. Upwash exploitation and downwash avoidance by flap phasing in ibis formation flight. *Nature*, 505(7483):399–402, 2014.

[68] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH*, 21:25–34, August 1987.

[69] RoboCup Small Size Robot League. Mixed Team Tournament, 2013. Online: `http://robocupssl.cpe.ku.ac.th/robocup2013:mixed_team_tournament`.

[70] S. B. Rosenthal, C. R. Twomey, A. T. Hartnett, H. S. Wu, and I. D. Couzin. Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion. *Proceedings of the National Academy of Sciences*, 112(15):4690–4695, 2015.

[71] P. Stone, G. Kaminka, S. Kraus, and J. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10)*, pages 1504–1509. AAAI Press, 2010.

[72] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[73] A. Strandburg-Peshkin, C. R. Twomey, N. W. F. Bode, A. B. Kao, Y. Katz, C. C. Ioannou, S. B. Rosenthal, C. J. Torney, H. S. Wu, S. A. Levin, and I. D. Couzin. Visual sensory networks and effective information transfer in animal groups. *Current Biology*, 23(17):R709–R711, 2013.

[74] D. Strombom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. Sumpter, and A. J. King. Solving the shepherding problem: Heuristics for herding autonomous, interacting agents. *Journal of The Royal Society Interface*, 11(100), 2014.

[75] H. Su, X. Wang, and Z. Lin. Flocking of multi-agents with a virtual leader. *IEEE Transactions on Automatic Control*, 54(2):293–307, 2009.

[76] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7(1):83–124, 1997.

[77] N. Tinbergen. Social behaviour in animals: With special reference to vertebrates. 1953.

[78] R. Tiwari, P. Jain, S. Butail, S. P. Baliyarasimhuni, and M. A. Goodrich. Effect of leader placement on robotic swarm control. In *Proceedings of the 2017 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'17)*, pages 1387–1394. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[79] A. Turgut, H. Celikkanat, F. Gokce, and E. Sahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2):97–120, 2008.

[80] T. Vicsek, A. Czirok, E. Ben-Jacob, I. Cohen, and O. Shochet. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6):1226–1229, 1995.

[81] M. Wilkerson-Jerde, F. Stonedahl, and U. Wilensky. NetLogo flocking vee formations model, 2009. Online: `http://ccl.northwestern.edu/netlogo/models/FlockingVeeFormations`.

[82] C.-H. Yu, J. Werfel, and R. Nagpal. Collective decision-making in multi-agent systems by implicit leadership. In *Proceedings of the 2010 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS'10)*, pages 1189–1196. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[83] M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9):1525–1540, 2011.