Design of Influencing Agents for Flocking in Low-Density Settings

A THESIS PRESENTED
BY
DANIEL Y. FU
TO
THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
ARTIUM BACCALAUREUS
IN THE SUBJECT OF
COMPUTER SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS

MARCH 2018

Thesis advisor: Professor Radhika Nagpal Daniel Y. Fu

Design of Influencing Agents for Flocking in Low-Density Settings

Abstract

Flocking is a coordinated collective behavior that results from local sensing between individual agents who have a tendency to orient towards each other. Flocking is common amongst animal groups and could also be useful in robotic swarms. In the interest of learning how to control flocking behavior, several pieces of recent work in the multiagent systems literature have explored the use of influencing agents for guiding flocking agents to face a target direction. However, the existing work in this domain has focused on simulation settings of small areas with toroidal shapes. In such settings, agent density is high, so interactions are common, and flock formation occurs easily. In our work, we study new environments with lower agent density, wherein interactions are more rare. We study the efficacy of placement strategies and influencing agent behaviors drawn from the literature, and find that the behaviors that have been shown to work well in high-density conditions tend to be much less effective in the environments we introduce. The source of this ineffectiveness is a tendency of influencing agents explored in prior work to face directions intended for maximal influence that actually separate the influencing agents from the flock. We find that in low-density conditions maintaining a connection to the flock is more important than rushing to orient towards the desired direction.

We use these insights to propose new influencing agent behaviors that overcome the difficulties posed by our new environments. The best influencing agents we identify act like normal members of the flock to achieve positions that allow for control, and then exert their influence. We dub this strategy "follow-then-influence."

We also tackle this problem by using genetic programming to evolve ad hoc behaviors for influencing agents. We use a hand-constructed domain-specific language and evolve populations in a small test environment, before testing the best candidates in larger scenarios. We find that the best genetic behaviors can do as well as our best hand-designed algorithms, since they can strike a balance between quickly turning their neighbors towards the goal direction while not losing influence by flying away from their neighbors.

Contents

Listing of figures

This is the dedication.

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

1

Introduction

Across nature, flocking behavior can be found in a variety of species, from flocks of birds, to herds of quadrupeds, schools of fish, or swarms of insects. In such species, groups exhibiting collective behavior emerge from simple, local rules [26]. An open question is whether externally-controlled influencing agents can be used to affect the behavior of flocks.

Previous work [7, 8, 9, 12, 11, 10, 6] has explored the use of influencing agents to guide flocking

agents to face a target direction in small and toroidal settings. In such settings, agent density is high, so interactions are common, and flock formation is rapid. In this work, we build upon past work by studying lower-density settings where interactions are rarer and flock formation is more difficult. As a result, influencing agent priorities must change in these settings to be successful. We tackle the problems introduced by these new settings by introducing new influencing agent strategies and by using genetic programming to automatically explore a wide range of the solution space.

Low-density settings are important to study because they capture dynamics in situations where flocking may not occur naturally, but where we might want to instigate flocking behavior; imagine a herd of buffalo that is currently grazing, or a spooked flock of birds where individual agents fail to coordinate. It may also have implications for coordination in low-density swarms of robotic multi-agent systems, where control may be imperfect, such as RoboBees [2]. More broadly, flocking has implications for consensus in animal groups [30, 27, 3] and in human social networks [17]; it has also been used to model multivariate timeseries and study human movement [22, 24]. In all these cases, agent density may vary greatly, so it is important to understand influencing agent dynamics in both low density and high density settings.

For our work, we draw heavily from a recent series of studies of this problem by Genter and Stone [7, 8, 9, 12, 11, 10, 6]. Genter and Stone study influencing agent placement and behavior in small, toroidal environments.* In such environments, agents are bound to a small space and have unlimited opportunities to re-enter the screen and join a flock; flock formation is rapid.

*In a toroidal environment, agents that exit the simulation space from one side immediately re-appear on the other side

We study this question in a more adverse environment by proposing two new test settings with lower agent density. In one setting, we keep the simulation space toroidal but increase the size of the space by several factors, greatly decreasing agent density. Flock formation is still provably guaranteed in this setting, but is much less rapid, so we study whether influencing agents can speed up flock formation. In the other setting, we make the simulation space non-toroidal but start the flocking agents in a circle in the center. Since this space is non-toroidal, flock formation is not guaranteed, so we study whether influencing agents can instigate flocking behavior by keeping the flocking agents in a pre-defined area, or move them all in a certain direction.

We find that in environments with low agent density, agent interactions are rare, so results from experiments in smaller settings do not translate well to larger settings. In particular, when agent interactions are rare, maintaining a connection to the flock becomes a key factor in the efficacy of influencing agent behaviors. As a result, simple local behaviors such as "face the goal direction" are often superior to more complex local behaviors that try to optimize for speed. We also experiment with a number of new behaviors and find that a multi-stage approach of "follow-then-influence" can be effective in certain situations. In this approach, influencing agents start out by obeying the flocking rules, embedding themselves inside small, naturally-forming flocks. After some time, the influencing agents start influencing their neighbors to face a given goal direction. Finally, we find that reasonable placement strategies are often interchangeable, and that more complex strategies often perform similarly to random strategies.

Next, we use genetic programming to generate different new behaviors. In order to build behaviors that work with variable numbers of neighbors, we evolve programs that are applied iteratively

to each neighbor using a functional programming construct called a fold. In a fold, a function is applied in turn to each item in a list, updating an accumulator value for each one. The final value of the computation is the value of the accumulator after each item has been folded over.

We define a simple language that performs arithmetic computations on scalar values and angles, and evolve different functions to use in the fold. To make the evolutionary process fast enough, we first evaluate different functions in a small environment, with six neighbors that start in a circle around a single influencing agent. We run four trials; in each trial, the six neighbors (and the influencing agent) are facing one of the cardinal directions, and the target direction is east. We measure the average angle offset from the goal direction of the flocking agents after 100 steps and take the average across the four trials as the fitness value for the proposed program. We then run the evolutionary algorithm on this fitness value and mutate the best candidates each generation.

To evaluate these behaviors in a wider context, we then test them in our two new settings as well. We find that the genetic behaviors can both encourage faster convergence in the short term while still maintaining influence over the long-term. As a result, they can perform as well as the best hand-constructed algorithms. However, they can also be difficult to understand and reason about. In particular, some genetic behaviors work very well when paired with certain global behaviors, but experience catastrophic failures when paired with others.

The main contributions of this work are:

- An investigation of two new low-density flocking settings, where flock formation is more difficult.
- The introduction of new placement strategies and influencing agent behaviors to adapt to the difficulties presented by these new settings.

- Analysis of the major differences in influencing agent priorities in low-density vs. high-density settings.
- A functional language to define ad hoc influencing agent behaviors.
- An evolutionary algorithm to evolve and evaluate candidate programs in this language.
- Analysis of the efficacy of the programs evolved by this algorithm and their strengths and shortcomings relative to hand-constructed algorithms.

The rest of this thesis is organized as follows: §2 introduces background and related work. §3 describes the formal flocking model and the capabilities of influencing agents, as well as the settings we wish to study. §4 discusses the placement strategies and behaviors that we study, as well as our strategy for evolving components of influencing agent behavior. In §5, we present our metrics and experimental frameworks, followed by the results from our experiments. Finally, we conclude and discuss future work in §6.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

2

Background and Related Work

2.1 Previous Work by Genter and Stone

Our work strongly builds upon the work of Genter and Stone, who have recently published a series of papers on the best way to influence an existing flock to change direction [12, 9, 8, 7, 11, 10]. This prior literature has studied a number of placement strategies and influencing agent behaviors, including questions of how best to join or leave a flock in real scenarios. Genter's PhD thesis also presents

results from simulations with a slightly different implementation of Reynold's flocking model, as well as physical experiments with these algorithms in a small RoboCup setting.

This work has studied influencing agents in a number of contexts, including convergence in a small setting and steering a small, dense flock around an obstacle. We are primarily interested in building upon the work from the former context, although the second context is still interesting. In their experiments, Genter and Stone place 180 flocking agents and 20 influencing agents on a 150 $\times$ 150 toroidal grid, and let agents influence each other if they are in a neighborhood radius of 20 units or less from each other. As a result, they simulate the dynamics of an extremely high-density flock. In our work, we introduce new settings to study dynamics in low-density flocks. We will introduce the formal flocking model and discuss Genter and Stone's placement strategies and algorithms in more detail in §3 and §4, but we will summarize Genter and Stone's key results here to help situate our results.

Genter and Stone introduce a number of behaviors for influencing agents and compare them against two baselines: *face* and *offset momentum*. In *face*, influencing agents simply face a predetermined goal direction at all times; in *offset momentum*, influencing agents look at their neighbors, calculate an average velocity vector, and choose a direction to offset that velocity from the predetermined goal direction. The most promising behavior they study is *one step lookahead*; in this behavior, influencing agents cycle through choices for every direction they can take, and simulate one step of their neighbors for every choice they take. The agents pick the choice that minimizes the average difference of their neighbors from the goal direction. Not surprisingly, *one step lookahead* beats the baseline behaviors in their experiments. We will see that this is not the case in low-density settings

and discuss the reasons behind this surprising shift in §5.

2.2 Other Related Work in Flocking

There is a rich history of work studying flocking, much of it concentrating on flocking models originally proposed by Reynolds [21] and, independently, Vicsek [29]. The classical Reynolds model identifies three properties necessary for flocking:

- Alignment: a tendency for flocking agents to steer towards the average heading of their neighbors
- Avoidance: a tendency for flocking agents to avoid collisions with their neighbors
- Cohesion: a tendency for flocking agents to steer towards the average position of their neighbors

Separately, Vicsek proposed a flocking model mathematically equivalent to the alignment portion of the Reynolds model. As a result, many subsequent studies that build off the Reynolds or Vicsek models, including the studies of Genter and Stone, only study the alignment portion.

Jadbadbaie et. al. [15] have studied this model from an analytical perspective. Two strong results from this work are that a group of Reynolds-Vicsek agents in a toroidal setting will eventually converge regardless of initial conditions, and that in the presence of a single agent with fixed orientation (analogous to a single influencing agent), all the agents will converge to that fixed agent's orientation. This provides important context for Genter and Stone's work and the work that we present here; when the setting is toroidal, convergence is guaranteed, so the interesting question how fast we can reach convergence.

Couzin et. al. [3] have studied the design of influencing agents for flocking as well, albeit with a slightly different model. They have proposed an influencing behavior wherein influencing agents adopt orientations "in between" their desired goal orientation and the orientations of their neighbors, in order to still influence their neighbors while not adopting orientations so extreme that they have no chance of being effective in the long term. In §4, we adapt this algorithm to the Reynolds-Vicsek flocking model to see how well it performs in our settings.

Han et. al. have [14] published a series of papers showing how to align a group of agents in the same direction. This literature has assumed a single influencing agent with infinite speed, and has used this property to construct a behavior that has the influencing agent fly around and correct the orientation of agents one at a time. The result is that the Reynolds-Vicsek agents all eventually converge to the target direction, but are not connected to each other. In our work, we limit the speed of influencing agents to be the same as the Reynolds-Vicsek agents to prevent the use of behaviors like this, and in hopes that our results will be more applicable to real applications; we suspect that influencing agents that act similarly to real birds will be more successful in real-world applications.

Su. et. al. [25] have also studied the question of flock formation and convergence, but have studied the question in the context of the Olfati-Saber flocking model [20]. This model assumes the existence of a single virtual leader that non-influencing agents know about. The virtual leader plays the role of an influencer here, but has special control over the other agents based on its status. In our work, we assume that influencing agents do not have any special interaction rules with Reynolds-Vicsek agents.

Other researchers have tackled this question with real flocking agents. Halloy et. al. [13] have

used robotic influencing agents to move cockroaches to areas they would otherwise avoid. Cockroaches display flocking behavior, but with very different models from the one that we study. In this case, Halloy et. al. have exploited the cockroaches' inability to differentiate between real cockroaches and the robotic influencing agents.

Vaughan et. al. [28] have used robotic influencing agents to herd a flock of ducks (on the ground) to a goal position in a small caged area. The approach here largely uses the robot agents to "push" the ducks from a distance, like a dog herding sheep. The dynamics in this case are very different from the models we study; when the ducks are on the ground, they can stand still, for instance, and the fence limits the ducks' behavior.

2.3 Genetic Programming

Genetic programming [16] uses evolutionary algorithms to explore the solution space of possible programs to optimize some metric. Common to all genetic programming techniques is some sort of genome that can be evolved over; in our work, this is a hand-designed domain-specific language that takes a list of neighbors as input and outputs an orientation for the influencing agent to face. We directly evolve the programs that define our influencing agents' behavior. Other choices for genome include neural networks or other data structures that can take the relevant input and generate a desired output; we do not explore these in our work.

The evolutionary algorithms themselves can vary in specifics, but they all require some mutation function and fitness function, and sometimes a crossover function. A mutation function takes a genome and "mutates" it in some way to produce another genome. In our work, this can happen by

changing the value of a constant or a function somewhere in the abstract syntax tree; in a genome defined by a neural network, this can take the shape of changing the weights between some number of nodes. The fitness function is used to evaluate individual genomes; each generation, the output from the fitness function is used to select some number of genomes to mutate or carry on into the next generation (survival of the fittest). Finally, some evolutionary algorithms, including ours, use a crossover function that takes two genomes and creates a new one from aspects of both (to simulate sexual reproduction).

We are not the first researchers to apply genetic algorithms to the domain of controlling agents in swarms; Dorigo and Trianni used genetic algorithms to evolve controllers for s-bots, swarming robots designed to self-assemble and move together as a swarm-bot [5]. Their problem formulation is slightly different than ours, however. In their project, they have complete control over every agent in the swarm; the goal of their genetic algorithm is to produce a program that can create collective behavior when run on every agent. In our work, we only have control over a limited number of influencing agents. In their work, Dorigo and Trianni use a neural network from sensors to the actuators of their robots as their genome. Although it would be interesting to apply this approach to our problem, we do not do so in this work.

Instead, we use genetic programming over a domain specific language designed with domain expertise. This is similar in approach to Sean Luke's use of genetic programming to build a team for the RoboCup97 competition [18]. One major difference is that our language is simpler than his, since the virtual soccer players are capable of much more complex behaviors than our flocking agents. As a result, his language contained many mroe primitives than the one we present in our

work. The execution model used by the RoboCup players was designed to simplify the work to be done by the evolved program without unnecessarily biasing it toward any particular strategy. We have similar goals when designing our own execution model and language.

Much work has been done on developing techniques for evolving programs in richer languages with complex constraints on which programs are valid [1]. Although we use functional programming ideas in our execution model, we keep our language simple, with only conditional expressions for control flow structures and only a single type. Exploring a richer language with a type system to reduce the number of useless programs our genetic algorithm searches would be very interesting, but we leave it as future work.

This is some random quote to start off the chapter.

Firstname lastname

3

Problem Description

3.1 Flocking Model

We use a simplified version of Reynold's Boid algorithm [21] to model the flock. In this simplified model, also proposed independently by Vicsek and collaborators [29], agents change their alignment at every step to be similar to the average alignment of other agents in their neighborhood. At each time step, each agent a_i moves with constant speed $s = 0.7$, has orientation $\vartheta_i(t)$, and position

$p_i(t) = (x_i(t), y_i(t))$. At timestep t, a_i updates its position based on its alignment: $x_i(t) = x_i(t-1) + s\cos(\vartheta_i(t))$ and $y_i(t) = y_i(t-1) - s\sin(\vartheta_i(t))$. At the same time, the agents change their orientation based on the alignments of neighboring agents. Let the neighbors $N_i(t)$ be the set of agents at time t that are within neighborhood radius r of a_i, not including a_i itself. At timestep t, each agent updates its orientation to turn towards the average of its neighbors' orientations:

$$\vartheta_i(t+1) = \vartheta_i(t) + \frac{1}{2}\frac{1}{|N_i(t)|}\Sigma_{a_j \in N_i(t)}\vartheta_j(t) - \vartheta_i(t),$$

The $\frac{1}{2}$ term denotes a "momentum" factor, wherein agents have a tendency to keep their current orientation.

3.2 Influencing Agents

We can change flock dynamics by introducing influencing agents that we control. We refer to non-influencing agents as Reynolds-Vicsek agents. We do not give the influencing agents any special control over Reynolds-Vicsek agents; we simply let the latter interact with influencing agents using the same local sensing rules as with any other agent. We also limit the influencing agents to have the same speed as Reynolds-Vicsek agents, both to help the influencing agents "blend in" in real applications and to conform to the Genter and Stone experiments. We let the influencing agents have a sensing radius of twice the normal neighborhood radius. This allows for more complex behaviors like *one step lookahead* without requiring a global view from the influencing agent. In some cases, the influencing agents can communicate with each other, but they do not need global GPS.

In order to study low-density dynamics, we introduce two new settings that are more adverse to flock formation; we call these new settings the *large* setting and the *herd* setting. In both these settings, we set the neighborhood radius to $r = 10$.

In the *large* setting, non-influencing agents are randomly placed in a toroidal 1000×1000 grid with random initial orientations. The larger grid size results in lower agent density; as a result, agents start out much farther away from other agents' neighborhoods, and interactions are much rarer. However, since the simulation space remains toroidal, convergence to a single flock is still provably guaranteed, so we are primarily interested in studying the length of time to convergence in this case [15]. In the *herd* setting, non-influencing agents are placed randomly in a circle of radius 500 whose origin lies at the center of a 5000×5000 non-toroidal grid. When agents reach the edge of the grid, they simply keep going; in this way, agents can get "lost" from the rest of the flock. As a result, convergence to a single flock is not guaranteed. Therefore, we are interested in studying how well influencing agents can keep the non-influencing agents from getting lost.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

4

Algorithms for Influencing Agents

There are two important components to influencing agent design: placement and behavior. Placement refers to the initial placement of influencing agents in the grid. There has been some prior work on the question of how to navigate the influencing agents to get to their initial positions [12], but in this work we simply place the influencing agents in their initial positions immediately. Behavior, on the other hand, refers to how the influencing agent changes its orientation over time to meet

its end goals.

In this chapter, we discuss placement strategies and hand-designed behaviors, as well as our strategy for evolving new behaviors.

4.1 Placement Strategies

In this work, we study a number of different placement strategies, shown in Figure 4.1. Except for slight modifications to make some of these strategies circular, all the strategies are drawn from the literature [12, 6]. We note that the question of how to maneuver influencing agents to reach the positions given by these placement strategies is important, but out of scope for this paper. For a discussion of this question, we refer the reader to Genter and Stone [11, 6].

We use three placement strategies for the *large* setting: *random*, *grid*, and *k-means*. The random placement strategy, as its name suggests, places influencing agents randomly throughout the grid. The grid placement strategy computes a square lattice on the grid and places influencing agents on the lattice points. This strategy ensures regular placement of influencing agents throughout the grid. The k-means placement strategy uses a k-means clustering algorithm on the positions of Reynolds-Viscek agents in the simulation space. This strategy finds a cluster for each influencing agent by setting k equal to the number of influencing agents, and then places an influencing agent at the center of each cluster.

We develop similar placement strategies for our *herd* setting, with some slight modifications. To adapt the strategies to a circular arrangement of agents, we define each strategy in terms of some radius r about an origin O, except for the *k-means* strategy, which remains the same. We modify the

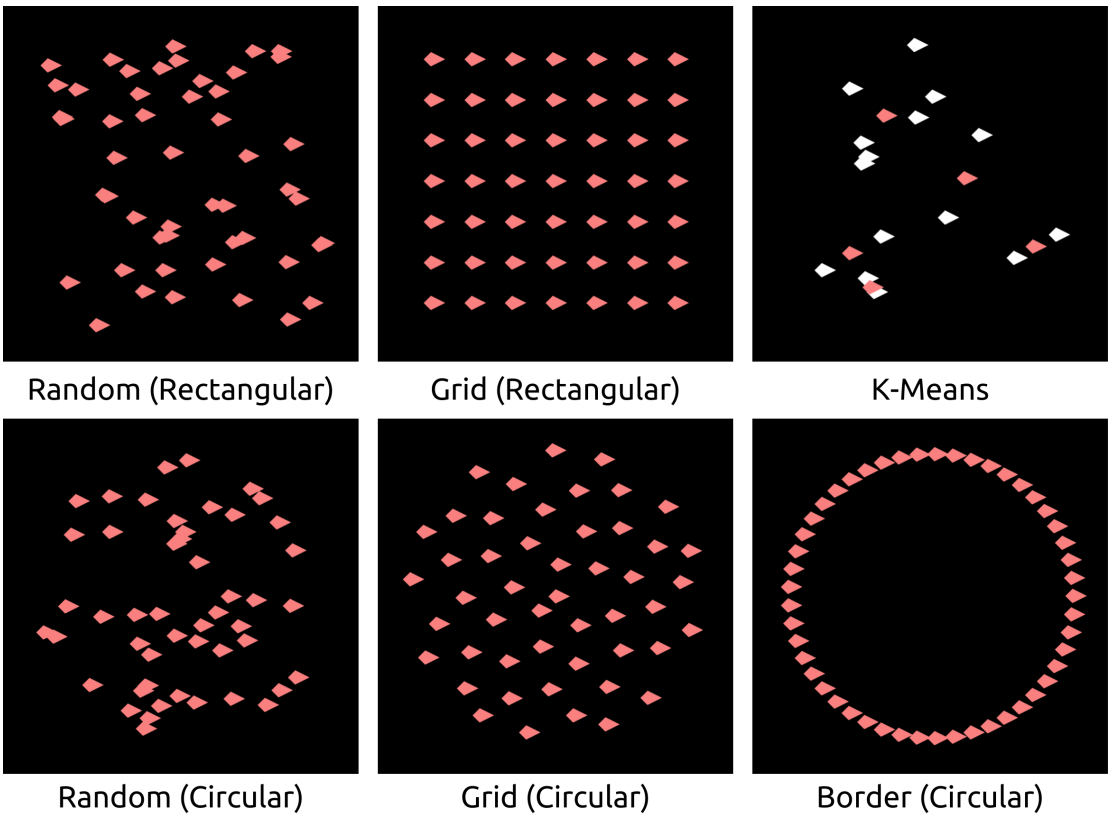


Figure 4.1: The different placement strategies we explore in this paper. Red agents are influencing agents, and white agents are Reynolds-Viscek agents. Note that k-means is the only placement strategy where the placement of influencing agents depends on placement of Reynolds-Viscek agents.

random placement strategy to randomly distribute agents within the circle of radius r about the origin O, instead of the entire simulation space. We adapt the grid placement strategy to a circular setting using a sunflower spiral [23]. In polar coordinates relative to O, the position of the n-th influencing agent in a sunflower spiral is given by $(c\sqrt{n}, \frac{2}{\varphi}n)$, where φ is the golden ratio, and c is a normalizing constant such that the last influencing agent has distance r from O. We also introduce a circle border placement strategy, inspired from the border strategies from [12]. This strategy places agents on the circumference of the circle of radius r around the origin O. We refer to the circular strategies as *circle-random*, *circle-grid*, and *circle-border*, respectively.

4.2 Behaviors

Once we have placed our influencing agents, we still need to design how they will work together to influence the flock. We call this aspect of the design influencing agent behaviors. In the present work we focus on decentralized "ad-hoc" algorithms for our influencing agents, since this class of algorithms has been the focus of the existing multiagent systems literature on this topic [6, 9, 10]. A summary of the behaviors we investigate is shown in Table 4.1.

4.2.1 Large Setting

For the *large* setting, we study four behaviors drawn from Genter and Stone [10, 11, 12], one behavior adapted from Couzin et. al. [3], and one new *multistep* behavior.

In previous work, Genter and Stone have introduced baseline behaviors *face* and *offset momentum*, as well as more sophisticated behaviors *one step lookahead* and *coordinated*. Each of these be-

Table 4.1: Summary of the hand-designed behaviors we investigate

Setting	Type	Name	Description
Large		*Face*	Face goal
		Offset Momentum	Offset average velocity
		One Step Lookahead	Simulate one step
		Coordinated	Pair off and coordinate
		Couzin	Averaging
		Multistep	*Follow-then-influence*
Herd	Net	*Face*	Face goal
		Offset Momentum	Offset average velocity
		One Step Lookahead	Simulate one step
		Coordinated	Pair off and coordinate
		Couzin	Averaging
	Stationary	*Circle*	Trace circle around agents
		Polygon	Trace polygon around agents
		Multicircle	*Follow-then-influence*

haviors aims to turn Reynolds-Vicsek agents to a pre-set goal angle ϑ^*. In *face*, influencing agents always face the angle ϑ^*. In *offset momentum*, influencing agents calculate the average velocity vector of the agents in their neighborhood, and take on a velocity vector that, when added to the average velocity vector, sums to the vector pointing in direction ϑ^*. In *one step lookahead*, each influencing agent cycles through different angles and simulates one step of each of its neighbors if it were to move in that angle. It adopts the angle that results in the smallest average difference in angle from ϑ^* among all its neighbors. Finally, in *coordinated*, each agent pairs with another and runs a one step lookahead to minimize the average difference in angle from ϑ^* among both their neighbors. For a more detailed explanation of these behaviors, especially the *coordinated* behavior, we direct the reader to Genter and Stone [10].

Couzin et. al. explored algorithms that would allow "informed agents" to steer an animal group [3], albeit with a different flocking model. These informed agents are analogous to our influenc-

ing agents. They proposed an algorithm where influencing agents chose an orientation in between the goal direction and the direction they would travel if they were regular flocking agents. This algorithm is parameterized by a value ω and works as follows. Consider an influencing agent with goal orientation ϑ^*; let this be represented by a unit vector $\vec{v}^*$. Suppose the influencing agent's orientation at time t is given by $\vartheta(t)$; let this be represented by a unit vector $\vec{d}(t)$. Let $\hat{\vartheta}(t+1)$ be the direction that the influencing agent would face if it were a normal flocking agent; again, let this by represented by a unit vector $\hat{\vec{d}}(t+1)$. Then influencing agent would have direction

$$\vec{d}(t+1) = \frac{\hat{\vec{d}}(t+1) + \omega\vec{v}^*}{|\hat{\vec{d}}(t+1) + \omega\vec{v}^*|}$$

at time $t+1$. $\omega = 0$ corresponds to a regular flocking agent, while very high values of ω grow closer to the *face* behavior. Couzin et. al. found that increasing ω can lead to more accurate convergence, but at the cost of potentially breaking up the flock.

Finally, the *multistep* behavior is a novel contribution and adopts what we call a "follow-then-influence" behavior. In the initial stage, influencing agents simply behave like normal Reynolds-Viscek agents; as a result, they easily join flocks and become distributed throughout the grid. At the same time, each influencing agent estimates how many Reynolds-Viscek agents are path-connected to it; here, we define two agents as being path-connected if there is a path between them, where edges are created by two agents being in each other's neighborhood. The astute reader will note that an accurate calculation requires a global view from every influencing agent, since paths may extend arbitrarily far away from the influencing agent. In our algorithm, we only consider Reynolds-

Viscek agents that are within the sensing radius of the influencing agents. Given their local estimates, the influencing agents compute a global sum of all their estimates; once that sum passes over some threshold T, the influencing agents calculate the average angle $\overline{\vartheta}$ among all the agents that are locally connected to influencing agents, and from there adopt the *face* behavior with goal direction $\overline{\vartheta}$. We choose $\overline{\vartheta}$ to minimize the amount that each influencing agent needs to turn its flock on average.

We also explore some variations on the *multistep* behavior; notice that once the sum of connected agents passes the threshold T, any of the other behaviors studied can be used to turn the Reynolds-Vicsek agents towards the final goal direction $\overline{\vartheta}$. In other words, the *multistep* behavior can be paired with any other behavior. We study these pairings to see how effective they are.

4.2.2 Herd Setting

For the *herd* setting, we divide our behaviors into two categories: *net* behaviors and *stationary* behaviors. As a reminder, in the *herd* setting, the simulation space is non-toroidal, and all the Reynolds-Viscek agents start in a circle in the center. In this setting, flock formation is not guaranteed, so we are interested in using influencing agents to instigate flocking behavior. There are two different choices we can make; we can either try to force the Reynolds-Viscek agents to stay in the center (*stationary* behaviors), or we can let the influencing agents direct the Reynolds-Viscek agents away from their initial starting position (*net* behaviors). Since all our agents have a constant speed, the former is much more difficult than the latter, so we must evaluate them separately.

For our *net* behaviors, we can use all the behaviors used in the *large* setting, except for the *multistep* behavior. Since the world is non-toroidal, it is not guaranteed that the number of connected

agents will ever pass the threshold T; in this case, the influencing agents would simply wander forever.

We study three *stationary* behaviors: *circle*, *polygon*, and *multicircle*. The *circle* and *polygon* behaviors have each influencing agent trace a circle or polygon around the origin. For placement strategies where influencing agents have different distances to the origin, the influencing agents simply trace circles and polygons of different radii.

The *multicircle* behavior is analogous to the *multistep* behavior from *large*. The influencing agents start out by circling around the origin and wait for Reynolds-Viscek agents to enter their neighborhood. Once they detect Reynolds-Viscek agents in their neighborhood, they adopt a "following" behavior where they act like Reynolds-Viscek agents to integrate into a small flock. They continue this following stage until reaching a final radius r_F, at which point they again adopt a circling behavior. In addition to building influence by following before influencing, this behavior also makes maintaining influence easier; since the final radius is larger than the original radius, the final path turns less sharply than if the influencing agents had stayed at their original radius. To the best of our knowledge, this is the first presentation of such a multi-stage behavior to induce circling behavior under the Reynolds-Vicsek model in the literature.

4.3 Evolving Behaviors

To evolve behaviors, we run genetic algorithms over genomes containing abstract syntax trees for a simple domain specific language we designed specifically for computing a new heading from a list of neighboring agents.

4.3.1 Domain Specific Language

All evolved behaviors need to be well-defined for any number of neighboring agents, so we choose an execution model that generalizes over the number of neighbors a influencing agent has. Every expression in our domain specific language defines a function from an accumulator value representing a heading and a neighbor agent to a new accumulator value. Starting with an initial accumulator value, the same function is applied to each of the influencing agent's neighbors in decreasing order of distance. The result of each application becomes the accumulator value fed to the next application, and the result of the final application is the heading the influencing agent will take in the next time step. In functional programming terms, we are evolving a function used to fold over a list of neighbors. The pseudocode for this fold is in listing 4.1.

```
acc = initial_acc
for neighbor in neighbors:
    acc = f(acc, neighbor)
return acc
```

Listing 4.1: A fold

The order in which neighbors are evaluated is meaningful, since the evolved functions are not necessarily commutative. We choose to evaluate them in decreasing order of distance to ensure that the results of programs that do not use the accumulator value are determined by the nearest neighbor of the influencing agent. If the influencing agent has no neighbors, its calculated heading

is equal to its initial accumulator value. A genome is comprised of an abstract syntax tree for the folding function and the initial accumulator value. The initial accumulator value is either the influencing agent's current heading or the goal direction, decided at random when an initial genome is created. An example of a hand-crafted genome is given in listing 4.2.

```
acc=current;
(add acc (mul influence heading))
```

Listing 4.2: A small genome that sets the agent's heading to the average of its neighbors' headings

By using a fold as our execution model, we can keep the syntax of the language very simple. Because the execution model handles looping over the list of neighbors, there is no need for the language itself to have syntax expressing loops or lists. Although allowing the language to have lists and loops would increase its expressiveness, we opted to keep the language simple while still expressive enough to allow the genetic algorithm to develop complex behaviors. We also chose to leave out other possible capabilities such as mechanisms for storing state other than the accumulator value and communicating with other influencing agents.

The expressions that make up the domain specific language are given in table 4.2. When evaluated, every expression in the language has a value in the range $[-1, 1]$, which is interpreted as an angle (in radians) by multiplying by π. All angles are relative to the influencing agent's current heading. Addition, subtraction, sine, and cosine operators treat their arguments as angles, but multiplication, exponentiation, arcsine, and arccosine treat them as scalar values. Division is purposefully excluded from the language because there is no meaningful way to enforce that its result is in the range $[-1, 1]$.

Exponentiation is modified such that the exponent is implicitly shifted into the range $[0, 2]$ to enforce the same invariant.

We determined what information should be made available about neighboring agents by drawing on our own intuition about what information would be valuable in designing an influencing behavior, then transforming it to be in the range $[-1, 1]$. For example, knowing the index of the current neighbor allows for creating a function that gives more weight to closer neighbors (those with higher indices); knowing the distance to the current neighbor combined with the conditional expression, gez, allows for creating programs that ignore neighbors that are too close or too far away; and the product of the influence variable and some other value can be added to the accumulator to calculate an average over all neighbors. Figure 4.2 shows how the direction, distance, heading, and goal variables relate to the position of a neighboring agent.

To keep the language simple, angular and scalar values are used interchangeably. Introducing a type system to keep them separate would greatly reduce the space of programs the genetic algorithm has to search through, at the expense of complicating the genetic operators, which would have to respect such a type system. We leave implementing such a type system as well as implementing a richer language as future work.

4.3.2 Evolutionary Algorithm

The two genetic operators we use in our genetic algorithm are the classic operators in genetic programming, crossover and mutation [4]. Crossover works by picking a random subtree from each of two abstract syntax trees and swapping them. Mutation picks a node at random from an abstract

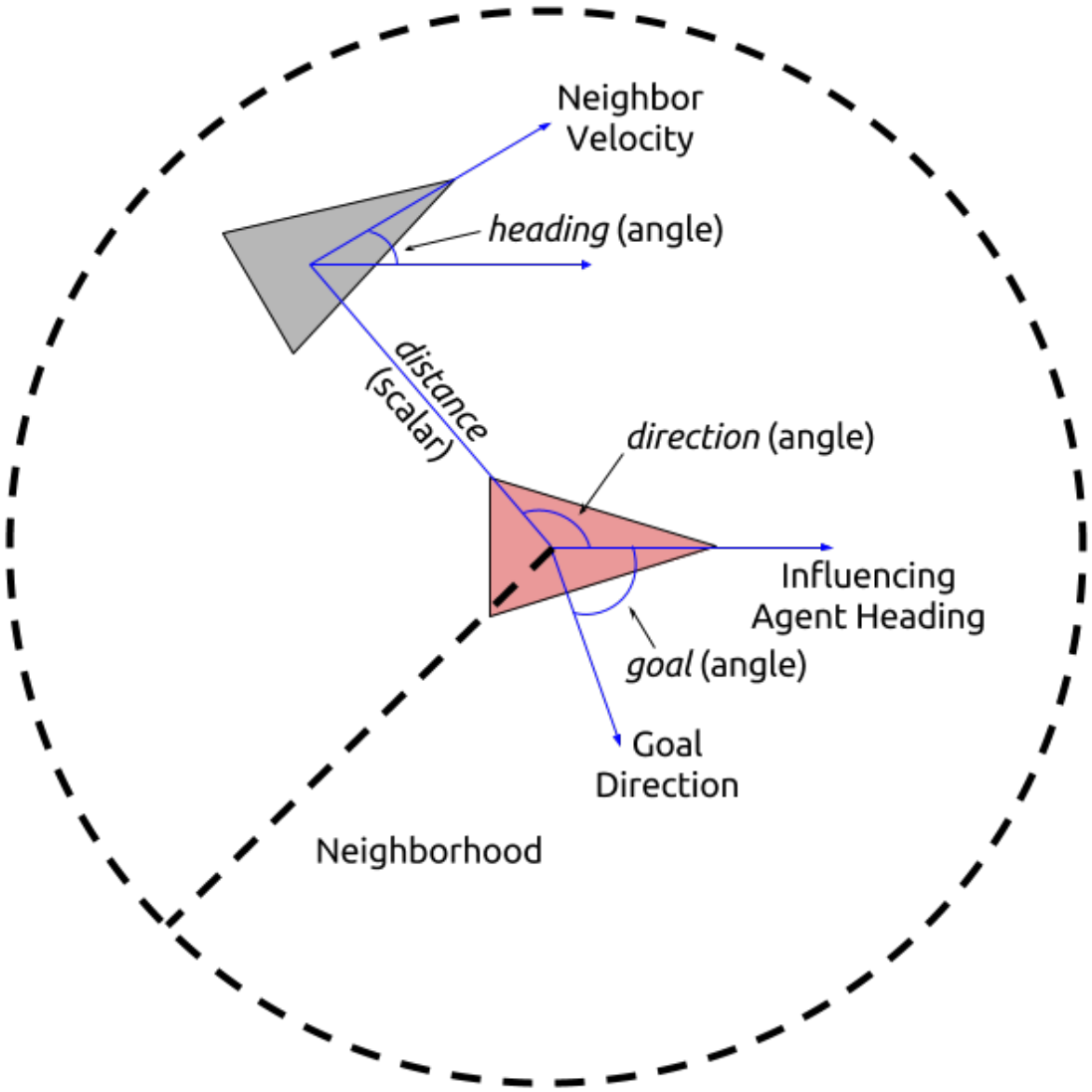


Figure 4.2: Some of the variables used in the domain-specific language.

Expression	Semantics
direction	angle toward neighbor
distance	distance to neighbor / sensing radius
heading	relative heading of neighbor
influence	1 / # of neighbors
index	index of neighbor / # of neighbors
goal	angle toward goal direction
acc	the accumulator value
f	some constant $f \in [-1, 1]$
(sin e)	$\sin(e \cdot \pi)$
(cos e)	$\cos(e \cdot \pi)$
(asin e)	$2 \cdot \text{asin}(e)/\pi$
(acos e)	$2 \cdot \text{acos}(e)/\pi - 1$
(add e_1 e_2)	$\text{wrap}_{[-1,1]}(e_1 + e_2)$
(sub e_1 e_2)	$\text{wrap}_{[-1,1]}(e_1 - e_2)$
(mul e_1 e_2)	$e_1 \cdot e_2$
(exp e_1 e_2)	$e_1^{e_2+1}$
(gez e_1 e_2 e_3)	if $e_1 >= 0$ then e_2 else e_3

Table 4.2: Language expressions

syntax tree and changes it into some new node, picked uniformly at random from the set of expression types available in the language. The children of the old node are made children of the new node in random order, although children may be lost if the new node has fewer children than the old node and new children may need to be randomly generated if the new node has more than the old node.

We start by randomly generating each member of the population. We evaluate each member with 6 Reynolds-Viscek agents in a small test setting, calculating the average absolute difference in angle of the Reynolds-Viscek agents from the goal direction after 100 steps. This gives a positive fitness value that we wish to optimize towards zero. In each generation, we choose the top N genomes

to use as a seed population for the next generation, where N is a hyperparameter set by the user. We add those N genomes into the new population. Then, while the size of the new population is smaller than the size of the old population, we repeat the following:

1. Flip a coin to decide whether to create a new member(s) by mutation or crossover.
2. If mutation, randomly select one genome from the seed population, mutate it, and add it to the new population.
3. If crossover, randomly select two genomes from the seed population, and use them to perform crossover. This creates two new offspring; add them both if there is room, otherwise randomly select one to add.

We experimented with a few variations on this algorithm over the course of this work. We started by choosing one individual in each generation to mutate, and randomly replaced one individual, chosen proportionally to fitness (keeping in mind that closer to 0 is better). However, it was difficult for this algorithm to favor better candidates, since fitness values were so close to each other. We tried switching to a rank-proportional replacement algorithm, but this approach had the opposite problem of distinguishing too finely between genomes with very similar fitness. We ultimately settled on our seed population approach as a compromise between these two forces. We also initially evolved genomes without the bias towards shorter genomes, but we found that this led to blow-out as soon as we added crossover.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

5

Evaluation

5.1 Experimental Setup

We extended the MASON simulator to run our experiments.[19] We used the default parameters for the Flocking simulation that is included with the MASON simulator, except without any randomness, cohesion, avoidance, or dead agents. We sampled metrics every 100 time steps, and ran all experiments for 100 trials.

5.1.1 No Influencing Agents

Previous literature compared new influencing agent behaviors with baseline influencing agent behaviors, but did not compare to settings with no influencing agents. In order to observe the marginal contribution of influencing agents in future experiments, we start our investigation of all three of our settings by studying flock formation in those environments without any influencing agents. We use two metrics to evaluate flock formation: average number of flocks formed and average proportion of lone agents at each time step.

In the *large* setting, we test on a toroidal 1000 $\times$ 1000 grid with agents that have a neighborhood radius of 10. In the *herd* setting, we use a non-toroidal 5000 $\times$ 5000 grid, and position a herd of agents that have neighborhood radius of 10 in a circle at the center of the grid with radius 500. For all settings, we vary the number N of agents from 50 to 300 in increments of 50. We run both sets of simulations for 6000 time steps.

5.1.2 Influencing Agents

Next, we evaluate the contributions of influencing agents. We first evaluate all our hand-designed algorithms. Then we use genetic programming to evolve some new behaviors, and compare these behaviors against a subset of the hand-designed algorithms to get a sense of how well they perform.

To evaluate the contributions of influencing agents in the *large* setting, we measure time to convergence. We define convergence as having half the Reynolds-Vicsek agents face the same direction, since full convergence takes much longer.

We test the *random*, *grid*, and *k-means* placement strategies, along with our full suite of behaviors in the *large* setting. We place 300 Reynolds-Vicsek agents on the grid and vary the number of influencing agents from 10 to 100 in intervals of 10.

To evaluate the contributions of influencing agents in the *herd* setting, we measure a slightly different metric. Since we have two qualitatively different categories of global behaviors (net behaviors vs. stationary behaviors), the number of agents facing the same direction is irrelevant, since the stationary behaviors rotate the agents around the origin (in fact, if the Reynolds-Vicsek agents are all facing the same goal direction, the stationary behavior has failed). Instead, we measure the number of Reynolds-Vicsek agents that are connected to influencing agents at 15000 time steps; at this point in time, all the agents have travelled out of the grid, and no new interactions occur. As a result, this is a measure of sustained influence over the Reynolds-Vicsek agents over time.

We examine the three circular placement strategies, *circle-border*, *circle-random*, and *circle-grid*, with two placement radii, *500* and *750*, along with the *k-means* placement strategy. We split our examination of global behaviors between the *net* behaviors (the same behaviors as used in the *large* setting, minus the *multistep* behavior) and three stationary behaviors - namely *circle*, *polygon*, and *multicircle*. We use a polygon with ten sides (a decagon) for our *polygon* experiments, and we vary the final radius for multicircle based on the initial placement radius. When the placement radius is *500*, we set the final radius to *900*; when the placement radius is *750*, we set the final radius to *1100*. Finally, we pair the all our local behaviors with all the global behaviors and placement strategies. We place 300 Reynolds-Vicsek agents on the grid and again vary the number of influencing agents from 10 to 100 in intervals of 10.

5.1.3 Evolutionary Experiments

The above experiments are important for final evaluation of our behaviors, but they are infeasible for short-term evaluation of genomes during the evolutionary process, since each trial of one of the large-scale experiments takes dozens of seconds to run. Furthermore, the random placement of the Reynolds-Vicsek agents creates high variability across multiple trials. Instead, we evaluate the genomes on a set of four small experiments, shown in Figure 5.1. In each of these experiments, the influencing agent is placed in the center of six Reynolds-Vicsek agents, each of which is at the edge of the influencing agent's neighborhood. All agents are initialized to face one of the cardinal directions, and we run the simulation for 100 steps. At the end, we measure the average absolute angle difference, in degrees, from the goal direction (east) across all six Reynolds-Vicsek agents. Lower values are better. Initially, we evaluated genomes during evolution by randomly placing Reynolds-Vicsek agents in a circle around the influencing agent. However, random placement resulted in high variability for the same genome between generations, and it became difficult for the algorithm to find better genomes. We ultimately settled on a deterministic test to both reduce this variability and to allow for memoization to make our algorithm run faster.

Finally, we add a small penalty for the number of nodes in the genome. This gives the following cost function:

$$\sum_{i \in \mathcal{A}} |\vartheta_i - \vartheta_{east}| + \alpha N_{nodes},$$

where $\mathcal{A}$ is the set of 6 Reynolds-Vicsek agents, N_{nodes} is the number of nodes in the genome, and α is a regularization factor. In our experiments, we set $\alpha = 0.1$ to allow development of complexity

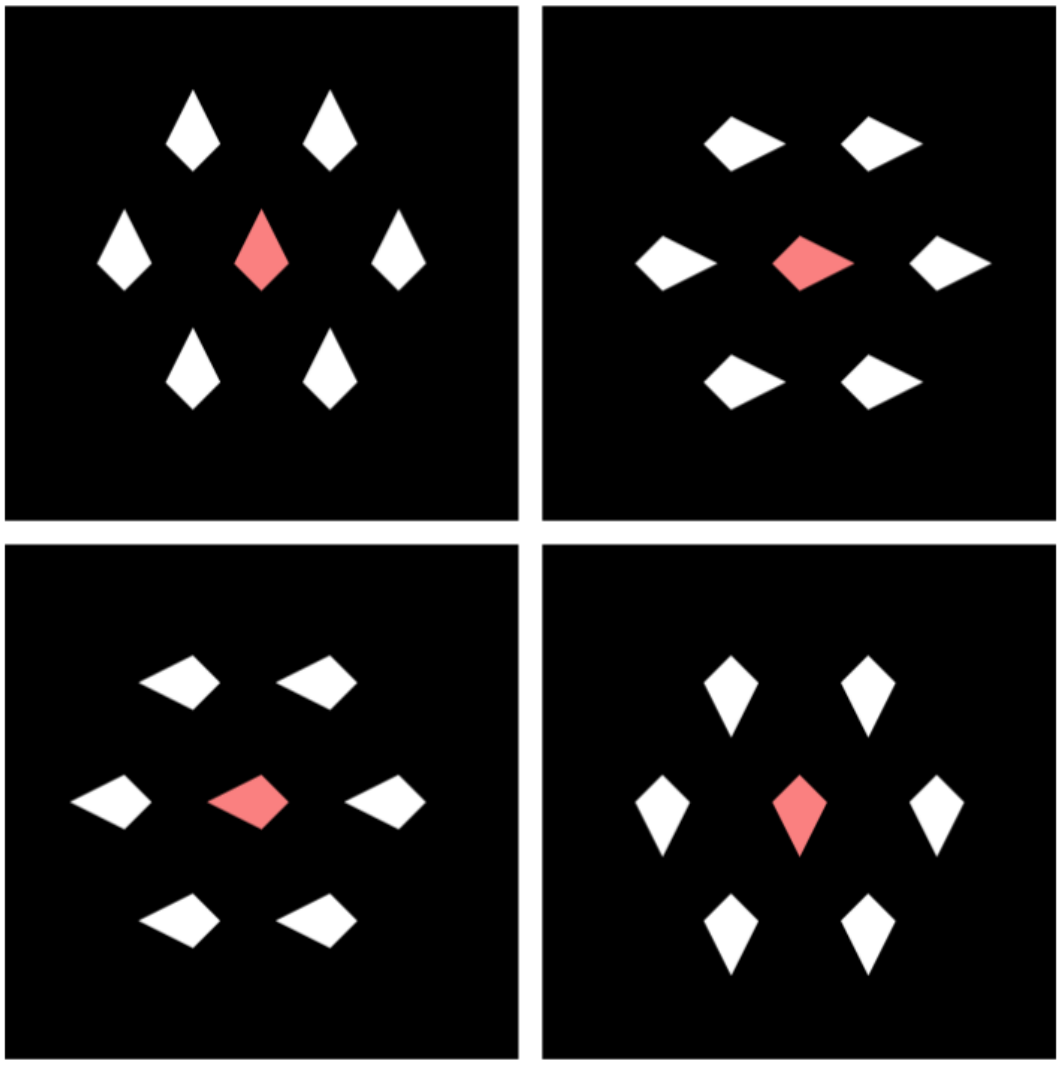


Figure 5.1: The four starting positions we use to evaluate genomes during evolution. We run the simulation for 100 steps and calculate the average angle offset from the goal direction across the six Reynolds-Vicsek agents. The goal direction is east.

but to prevent blow-out.

5.2 Results

5.2.1 No Influencing Agents

First, we briefly characterize the flocking behavior of a group of Reynolds-Vicsek agents without influencing agents in the *large* and *herd* settings. We measure the number of clusters of agents that are path-connected and facing the same direction; each of these clusters forms a small flock. We also measure the number of lone agents (the number of agents with no neighbors). Figure 5.2 shows graphs of these values over time for our two settings.

In the *large* setting, there are two qualitative stages of convergence: initial flock formation and flock unification. In the first stage, individual agents collide with each other and form small flocks. In the second stage, these small flocks that formed collide with one another and join together to form larger flocks. In Figure 5.2, the first stage is represented by the initial increase in the average number of flocks, and the second stage is represented by the following decrease in the average number of flocks. This behavior is reflected in the continually decreasing number of lone agents; since the number of lone agents continues to decrease over time, we know that the decrease in the total number of flocks is due to flock convergence. Note that when there are more total agents, the absolute number of lone agents decreases faster and reaches a similar value to the other cases by the end of the simulation. In other words, the *ratio* of lone agents to total agents hits a lower value when there are more agents, but the final absolute number of total lone agents is still similar to the other

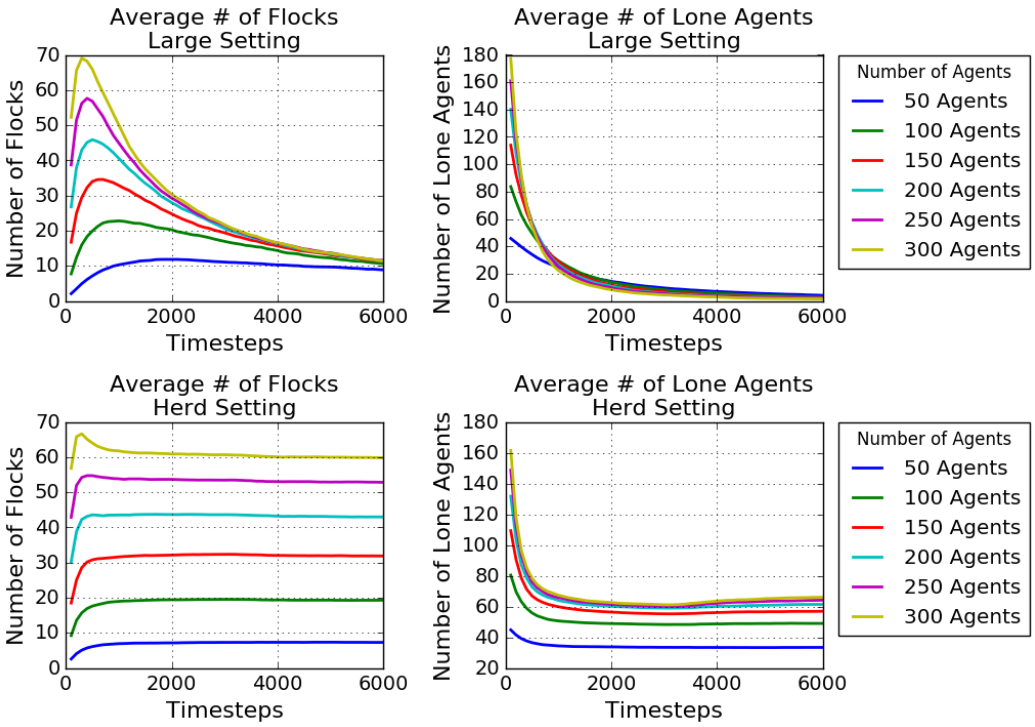


Figure 5.2: Average flock counts and lone agent counts over time for the *large* and *herd* settings with no influencing agents, varying the number of Reynolds-Vicsek agents.

cases.

The two stages of convergence also occur somewhat in the *herd* setting, but the second stage is cut off by the non-toroidal nature of the setup. As flocks leave the starting area, the chances of interacting with other flocks vastly decreases, so most of the flocks formed from the first stage never end up merging with other flocks. This is reflected in the plateaus of both the total number of flocks and the total number of lone agents. Some small artifacts in the graphs are worth mentioning; since the agents start off in a much smaller area than in the *large* setting, many of the agents start out with a non-zero number of neighbors. This causes the initial value of the average number of flocks to be non-zero, and the average number of lone agents to be less than the total number of agents.

5.2.2 Influencing Agents in the Large Setting

Next, we report on the efficacy of the hand-designed behaviors in the *large* setting. The average times for 50% convergence with different placement strategies and our primary six behaviors are shown in Figure 5.3. We show graphs for 50 influencing agents only, since the trends for the other numbers of influencing agents were similar (the major difference being that when there are more influencing agents, convergence happens faster). The results for the *Couzin* behavior are shown at $\omega = 0.5$; this value was experimentally shown by Couzin et. al. to be the highest value at this proportion of influencing agent that would not consistently break up the flock. Note that smaller is better in these graphs.

The most immediately striking finding is that, in our more adverse setting, the *one step lookahead* and *coordinated* behaviors significantly underperform the "baseline" *face* and *offset momentum*

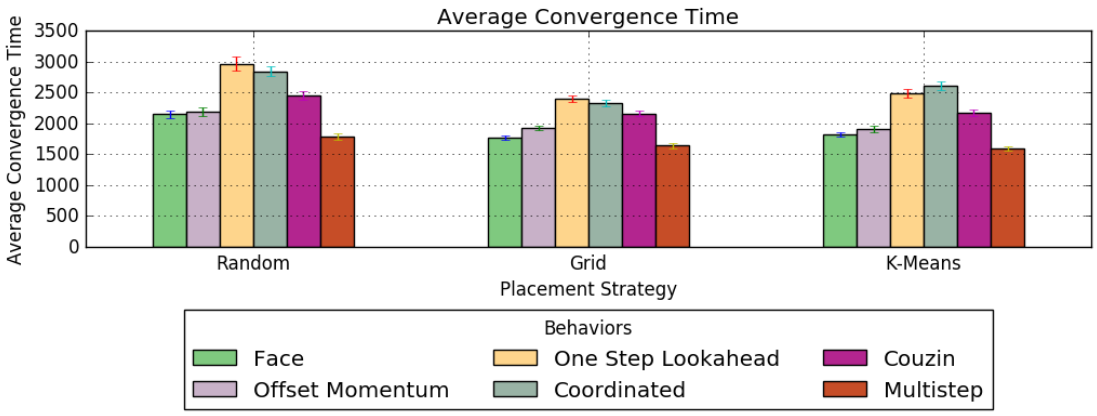


Figure 5.3: Average times to 50% convergence for 300 Reynolds-Vicsek agents with 50 influencing agents in the *large* setting under different placement strategies and behaviors. Smaller is better. Error bars show standard error of the mean.

behaviors, irrespective of placement strategy. This is an opposite result from Genter and Stone's findings on smaller simulation spaces [10, 6], which found that *one step lookahead* and *coordinated* outperform *face* and *offset momentum*. This finding is also rather counterintuitive; why should the "smarter" behaviors underperform the simpler behaviors?

The answer is that, when agent interactions are rare, it is more important for influencing agents to *maintain influence* than it is for them to quickly change the direction of neighboring Reynolds-Vicsek agents. The *one step lookahead* and *coordinated* behaviors underperform here because they tend to send influencing agents away from neighboring agents. An example of this phenomenon is shown in Figure 5.4. The influencing agent, shown in black, adopts an orientation that turns neighboring Reynolds-Vicsek agents towards the goal direction. Even though this does turn Reynolds-Vicsek agents towards the goal direction, it cannot successfully turn all the agents in a single step; as a result, it must maintain that orientation for future steps. However, as long as the neighboring

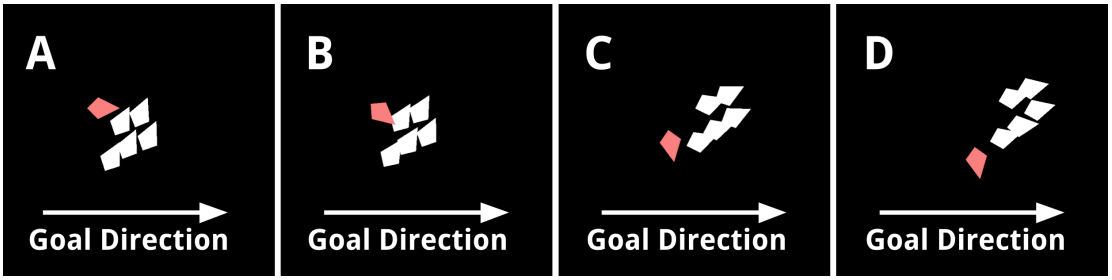


Figure 5.4: An example of an influencing agent losing influence under the *one step lookahead* behavior. The influencing agent is shown in red, and the Reynolds-Vicsek agents are shown in white. In A, the influencing agent first encounters the flock of Reynolds-Vicsek agents. In B-D, the influencing agent takes on directions that are oriented away from the goal direction to try to rapidly influence the Reynolds-Vicsek agents. This changes the orientation of the Reynolds-Vicsek agents, but the influencing agent has started to travel away from the flock by D.

agents are not facing the goal direction, the influencing agent's chosen orientation takes it away from the center of the flock of Reynolds-Vicsek agents, causing it to lose influence. Once the influencing agent has lost influence, it is difficult to catch up to the same flock, since influencing agents travel at the same speed* as Reynolds-Vicsek agents. As a result, the influencing agent is not actively influencing the direction of any Reynolds-Vicsek agents until it encounters another group of Reynolds-Vicsek agents.

Note that this effect also happens on a smaller simulation space, but it is not nearly as pronounced; when interactions are very frequent, influencing agents that have lost influence can find another group of Reynolds-Vicsek agents very quickly. As a result, the gains from the smarter local algorithm still outweigh the negative effects from losing influence.

However, it is important to strike a balance between not losing influence and turning Reynolds-

* There are some approaches which remove this speed constraint from influencing agents [14]. However, this allows for unrealistic behaviors wherein influencing agents travel to one Reynolds-Vicsek agent and a time and change the direction of the individual Reynolds-Vicsek agent before moving on to the next one. This results in Reynolds-Vicsek agents that are all facing the same direction, but that are often not path-connected.

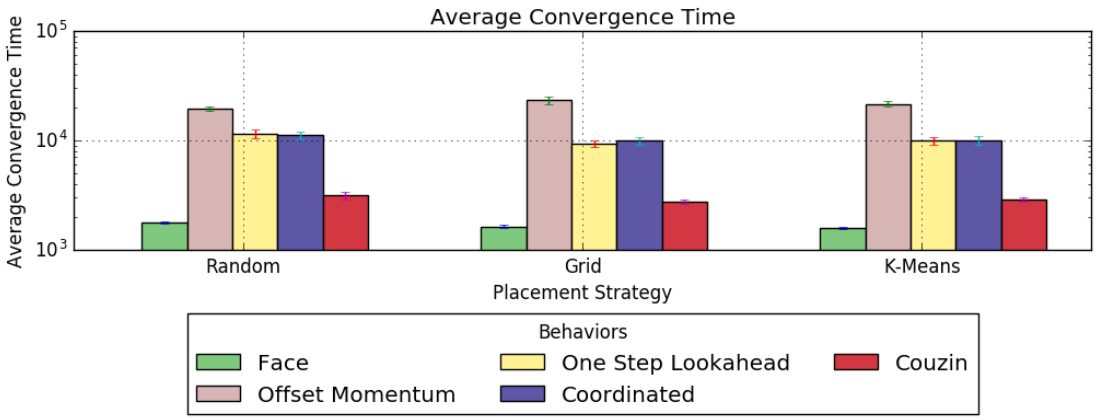


Figure 5.5: Average times to 50% convergence for 300 Reynolds-Vicsek agents with 50 influencing agents in the *large* setting under variations of the multistep behavior. Smaller is better. Error bars show standard error of the mean.

Vicsek agents too slowly. The *Couzin* algorithm is adept at maintaining influence, but slightly underperforms the *face* and *offset momentum* behaviors in measuring time to convergence, because it simply takes longer for influencing agents to turn the Reynolds-Vicsek agents.

The *multistep* behavior does not suffer from the same problem; it can both maintain influence and influence Reynolds-Vicsek agents and so outperforms all the other behaviors by a couple hundred steps (less so under the *grid* placement strategy).

When the *multistep* behavior is paired with the other behaviors, however, the effects of losing influence are magnified. Figure 5.5 shows variations on the *multistep* behavior, wherein influencing agents adopt different behaviors after the number of Reynolds-Vicsek agents under control passes T. Note that the variations that pair the *multistep* behavior with the *offset momentum*, *one step lookahead*, and *coordinated* behaviors perform almost an order of magnitude worse than the vanilla *multistep* behavior. What is the root cause of this difference? The *multistep* behavior starts out by creating many local flocks, some of which have influencing agents in them. When interactions are rare, the

offset momentum, *one step lookahead*, and *coordinated* behaviors have difficulties changing the orientation of existing flocks quickly before losing influence. As a result, the *multistep* behavior takes an order of magnitude longer to reach convergence when paired with the other local behaviors. This does not happen with the *Couzin* variation of the *multistep* behavior, since the *Couzin* variation can maintain influence (note that convergence still takes slightly longer, however).

Finally, we note that the effect of placement behaviors on convergence time are almost non-existent. When the density is lower, there is a much smaller chance that any influencing agent will start out with more than one Reynolds-Vicsek agent in its neighborhood, even with the *k-means* placement behavior. As a result, even the best clustering approach is almost the same as starting out randomly or in a grid.

5.2.3 Influencing Agents in the Herd Setting

Next, we evaluate results for our experiments in the *herd* setting. In many cases, measuring the number of agents facing the same direction is not interesting here, since it is impossible to keep Reynolds-Vicsek agents in one place if they are facing the same direction. Instead, we exclusively measure the number of Reynolds-Vicsek agents that are path-connected to influencing agents and facing the same direction as the influencing agent. This is a measure of "control" of the Reynolds-Vicsek agents. The average number of agents in such local flocks after 15000 time steps is given in Figure 5.6 for both the net and stationary behaviors. We find that the net behavior vastly outperforms any of the stationary behaviors. However, there may be environments in reality for which the net behavior is not applicable (suppose it is strictly necessary to keep a flock in one place, for instance). Thus, we

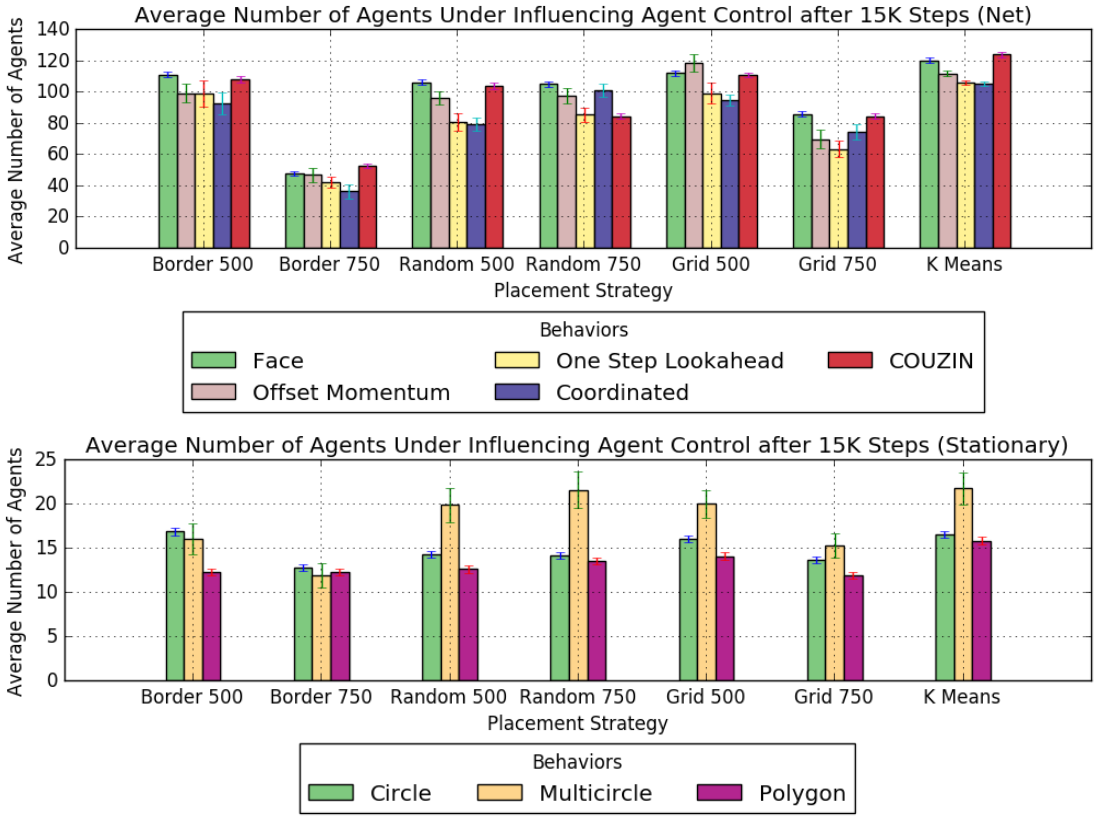


Figure 5.6: Average number of agents under influencing agent control after 15000 steps with 300 Reynolds-Vicsek agents and 50 influencing agents in the *herd* setting under for various placement strategies and influencing agent behaviors. The net behavior moves the influencing agents and their Reynolds-Vicsek agents off-screen, while the stationary behaviors keep the influencing agents near the goal area using some sort of circling technique. Larger is better. Error bars represent standard error of the mean.

analyze the net behaviors separately from the stationary behaviors.

Net

Again, we find that the *face* behavior tends to outperform the *offset momentum*, *one step lookahead*, and *coordinated* behaviors. Again, we attribute this to the tendency of the *offset momentum*, *one step lookahead*, and *coordinated* behaviors to lose influence over time. We note that the effect is not as pronounced here as in our *large* experiments, since each influencing agent has to control fewer agents. Note that the *face* behavior no longer outperforms the *Couzin* behavior, except with one placement strategy. Because the metric has changed to sustained influence over time, it no longer matters that the *Couzin* behavior turns the Reynolds-Vicsek agents slower.

In contrast to the *large* experiments, we do find that the placement strategy has a major effect on the efficacy of the net behavior. Again, this has to do with density of influencing agents. For example, notice that *Border 750* (place the influencing agents in a circle about the origin with radius 750) vastly underperforms the other placement strategies. The larger radius results in a lower density of influencing agents, so a greater number of Reynolds-Vicsek agents slip through the "holes" in the net. Furthermore, by the time the Reynolds-Vicsek agents reach the border, they have already formed flocks, and it is more difficult for the influencing agents to point them in the right direction. This effect is less pronounced for *Grid 750* and almost non-existant for *Random 750*, since these strategies place influencing agents within the circle, and not simply along its circumference. As a result, the Reynolds-Vicsek agents still encounter influencing agents before reaching the circumference of the circle.

Finally, we note that *k-means* outperforms all other placement strategies by a few agents. Again, the main driving factor behind this is agent density. When an influencing agent starts out in a clustered area, it has at least one other Reynolds-Vicsek agent in its neighborhood. As a result, its effective area of influence is slightly larger than with the other placement strategies. This helps it pick up more Reynolds-Vicsek agents in the net.

Stationary

For the stationary behaviors, we find that the *multicircle* behavior slightly underperforms the *circle* behavior when paired with the *Border* placement strategies; slightly overperforms when paired with the *k-means*, *Random*, and *Grid 500* placement strategies; and performs the same as *circle* in the *Grid 750* strategy. What drives these trends? Once *multicircle* reaches the final stage, it is tracing a larger circle than the *circle* behavior traces on its own. As a result, it is easier to maintain influence and turn the Reynolds-Vicsek agents over time in the final stage. Before that, however, the influencing agents are in a following stage. When the influencing agents start out inside the circle, they have more time to infiltrate small flocks of Reynolds-Vicsek agents and induce a circling behavior in the final stage.

Finally, we note that *Border 750* is the worst placement strategy, for reasons similar to the reasons for the net behaviors, and *polygon* tends to underperform or match the performance of *circle*. This tells us that adopting occasional sharper turns can sometimes be harmful, but not always.

5.2.4 Evolutionary Experiments

Next, we report on the results of our evolutionary experiments. We ran a total of 5 evolutionary runs. The first four had population size 25 and ran for 50 generations, while the last one had population size 100 and ran for 300 generations. For the first four runs, we selected the top 10 genomes to seed the next generation. For the last run, we selected the top 20 genomes. As an optimization, we memoized genomes from generation to generation and re-used results where possible. We allowed for duplicate members in the population. The average angle difference across the population and the angle difference of the best genome in each generation are shown in Figure 5.7. We also show performances of *face*, *offset momentum*, *one step lookahead*, and *Couzin* for comparison. We do now show *coordinated*, since *coordinated* is equivalent to *one step lookahead* when there is only a single influencing agent.

In each run, the population converged to one or two best genomes that remained in the seed population for every generation. These best genomes consistently outperformed the hand-constructed local behaviors taken from the literature. However, the average angle difference across the population stayed relatively high throughout the entire simulation, since the majority of the population were the results of random mutation or crossover from the seed population. In many cases, the new children perform very poorly, driving the average angle difference up. In the last run, with a population of size 100, there is a sharp drop in the average angle difference around generation 160. At this point in the evolutionary run, a single genome out-competed all the others. As a result, all subsequent populations were seeded by multiple copies of a single genome, and mutations/crossover

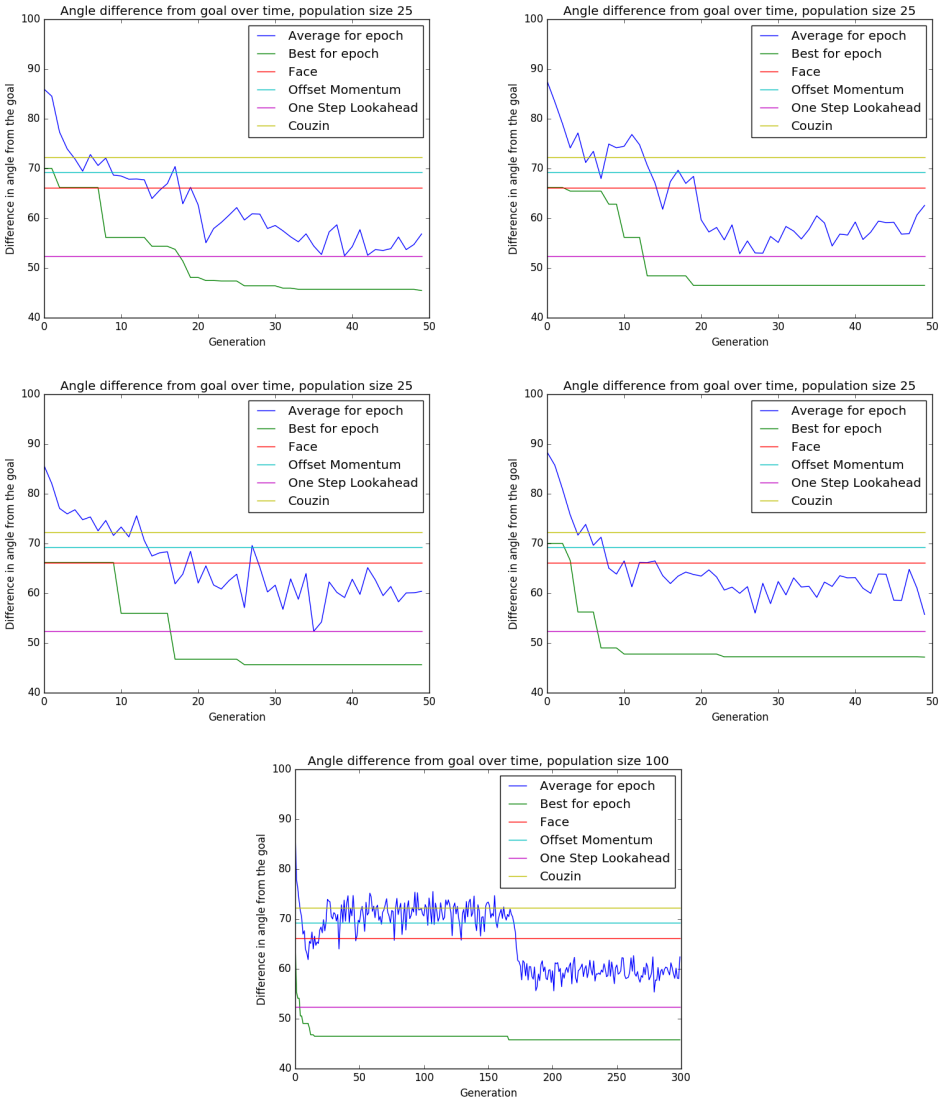


Figure 5.7: Average population fitness and fitness of the best candidate in the population over generations for a five different runs of the genetic algorithm. The first four runs had population size 25 and ran for 50 generations, while the last run has population size 100 and ran for 300 generations. Fitnesses of *face*, *offset momentum*, *one step lookahead*, and *Couzin* provided for comparison.

performed on this seed population were relatively stable.

From these five evolutionary runs, we selected 7 for further analysis. These are shown in Table 5.1 These genomes bear a few similarities to each other; they all initialize the accumulator to the goal

Name	Genome
G1	acc=goal; (mul acc (gez (cos heading) (exp (sin heading) distance) (sin goal)))
G2	acc=goal; (mul influence (mul influence goal))
G3	acc=goal; (sub acc (mul -0.26787718991714 44 goal))
G4	acc=goal; (mul (mul acc (cos heading)) distance)
G5	acc=goal; (mul distance (mul (sub (sub goal 0.2801770623593902) heading) (mul (sub acc heading) goal)))
G6	acc=goal; (mul distance (mul (sin influence) (mul (sub (sub goal 0.2801770623593902) heading) (mul (sub (sub goal 0.2801770623593902) heading) goal))))
G7	acc=goal; (mul goal (gez (cos acc) 0.19745673470999736 goal))

Table 5.1: Genomes from evolution

direction, for example. Beyond that, they share very little in common. Some genomes (G1, G3, G5, G7) use the accumulator, while the others do not; some are short and easy to parse (G2 is equivalent to $\frac{goal}{N^2}$, where N is the number of neighbors), while others are much more complicated.

Although they may be very different in expression, they are very similar to each other in execution. When all the agents are already facing east, these behaviors all have the influencing agent face east as well. In addition, they pick up some gains from the north and south starting conditions by initially picking up more neighbors than the baseline conditions. They pick up some major gains in the west case, where they intersect a few of the neighbors to their east side and lead them towards the target direction over the course of the next 100 steps. A few screenshots of this process are shown

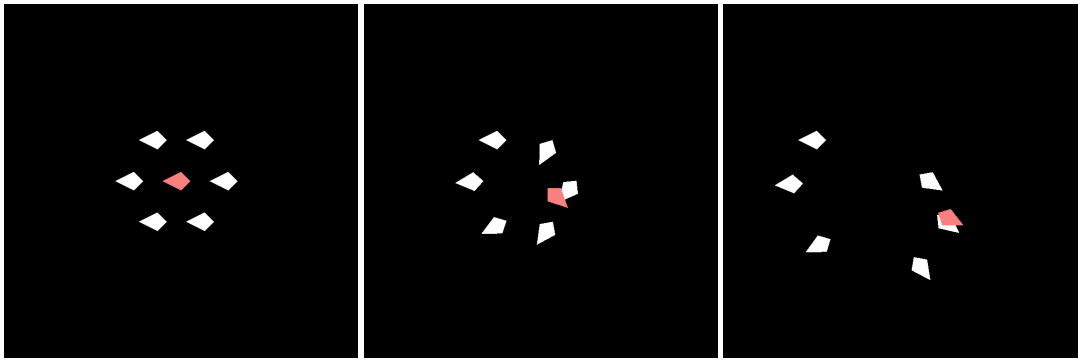


Figure 5.8: Screenshots of the G1 local behavior when trying to turn 6 Reynolds-Vicsek agents 180 degrees to the east. Instead of trying to turn all 6 neighbors at once, as the *one step lookahead* behavior does, the genetic behavior ``gives up'' on some neighbors and just tries to turn the neighbors that are behind it.

in Figure 5.8. The other genetic processes behave very similarly to the behavior shown in the screenshots, with slight variations (i.e., turning north instead of south, etc).

This behavior is very interesting scientifically for the differences between it and *one step lookahead*. In particular, the genetic behaviors immediately turn towards the Reynolds-Vicsek agents behind them, and ignore the agents in front of them. There are two major insights we can glean from this behavior. First, when the influencing agents focus on the agents behind them, they can successfully maintain influence over time instead of flying away from the flock as they try to turn their neighbors towards the goal direction. In other words, the genetic behaviors implicitly value influence over rapid convergence. We hypothesize that this arises because evaluation happens after 100 steps and not after 1 step, as with *one step lookahead*, but more work is needed to answer this definitively.

Second, the influencing agents ignore the Reynolds-Vicsek agents in front of them. There are a few reasons why this is beneficial. Since the influencing agents have limited speed, they can never

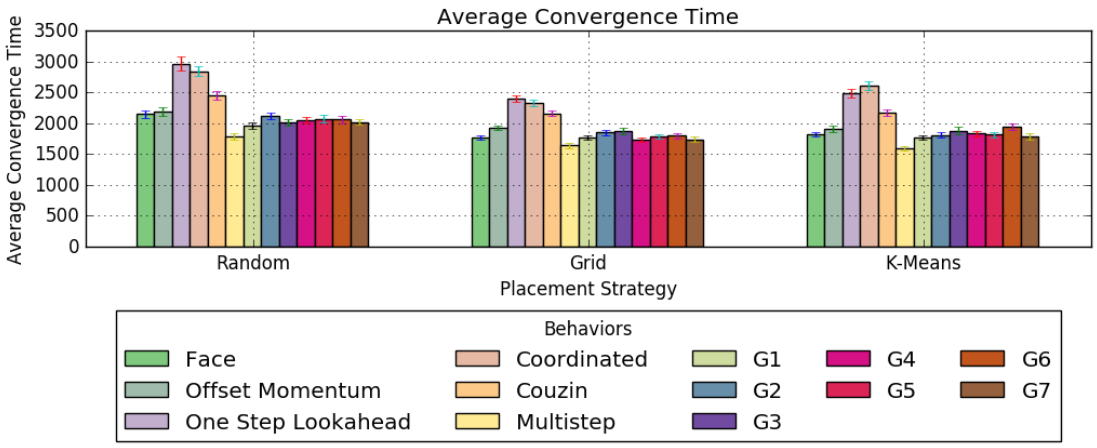


Figure 5.9: Average times to 50% convergence for 300 Reynolds-Vicsek agents with 50 influencing agents in the *large* setting under different placement strategies, paired with our hand-designed behaviors and the selected genetic behaviors. Smaller is better. Error bars show standard error of the mean.

"catch up" to the agents directly in front of them. The *one step lookahead* algorithm does not take this into account; as long as a neighbor is in range, its heading is factored into the calculation, no matter whether the influencing agent can ever reach the neighbor. Furthermore, it is much easier to lose influence over forward neighbors, since any change in direction necessarily means that the influencing agent falls behind. Thus, the Reynolds-Vicsek agents in front are somewhat of a lost cause, and it is beneficial to ignore them.

5.2.5 Influencing Agents in the Large and Herd Settings

Next, we verify that the genetic algorithms are still effective when used in our larger settings. For example, *one step lookahead* outperforms *face* in the small-scale evaluations we use during evolution, but underperforms *face* in simulations in the *large* setting. The results are shown in Figure 5.9.

We find that these genetic behaviors all perform on par with, or slightly better than, the *face* be-

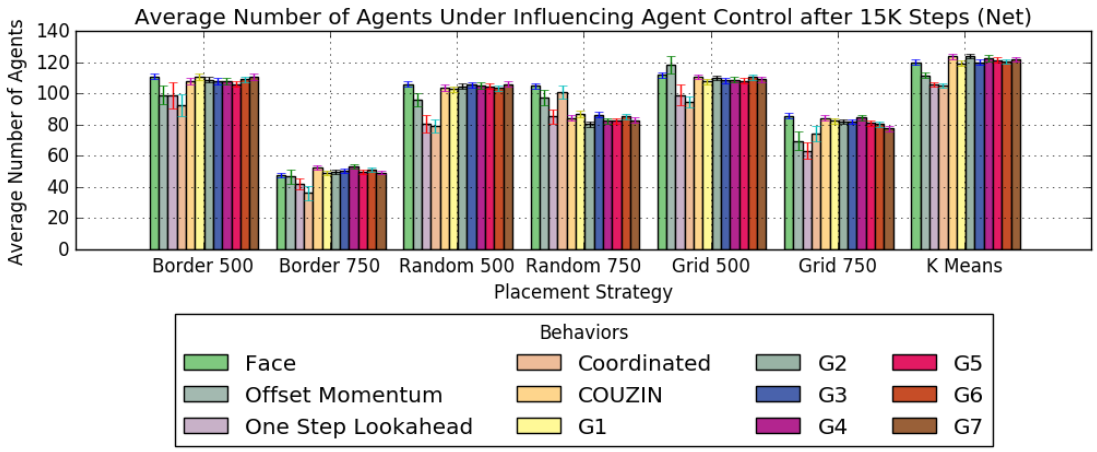


Figure 5.10: Average number of agents under influencing agent control after 15000 steps with 300 Reynolds-Vicsek agents and 50 influencing agents in the *herd* setting under for various placement strategies and behaviors, including the genetic algorithms. Larger is better. Error bars show standard error of hte mean.

havior. They all strongly outperform the *one step lookahead* and *coordinated* behaviors, and slightly outperform the *offset momentum* behavior. None perform as well as the *multistep* behavior, then *G1* comes close.

Figure 5.10 shows the performance of the genetic algorithms when used in the *net* case of the *herd* setting. Again, across the board, the genetic behaviors perform as well as *face* or *Couzin*, and outperform *offset momentum*, *one step lookahead*, and *coordinated*. The only exception is in the *Random 750* placement strategy, where all the genetic behaviors do worse. We are not sure what drives this effect, but we note that the *Couzin* strategy also performs poorly with this placement strategy; these results may be related in some way.

6

Conclusion

We have studied the problem of controlling flocks using influencing agents under two new, more adversarial environments with lower agent density, and have introduced some novel behaviors and placement strategies for these settings. Besides these new algorithms, we have found that, in low-density environments it is more important for influencing agents to *maintain influence* than it is for them to rapidly turn their neighbors towards the correct destination. As a result, earlier results

from smaller simulation environments often do not hold in the environments we introduce. We have found that a multistage approach that first embeds influencing agents in small flocks before attempting to steer these flocks to the goal direction can be effective in addressing some of these shortcomings.

We have also presented an evolutionary algorithm that evolves new genetic algorithms for these influencing agents. We evolved on a genome based on a hand-constructed domain-specific language and found that many of the evolutionary algorithms incorporated non-trivial insights about the best way to be successful in these settings. However, we also found that the evolved behaviors could be difficult to parse.

Future Work

There are a number of promising future directions that we could take from this work. We could explore the design space of genetic algorithms more deeply by evolving behaviors over different genomes, such as neural networks. We could also co-evolve different aspects of influencing agent behavior to help agents co-ordinate in previously unexpected ways; for example, we could introduce some form of communication between influencing agents, or co-evolve behaviors with placement strategies.

Future work could also explore the design space of placement strategies and agent behaviors by applying machine learning techniques to this problem; reinforcement learning seems to be a particularly intriguing way to develop new behaviors for influencing agents. We would also like to explore the question of how to aggregate small flocks into one larger flock. Almost all of our behaviors re-

sult in multiple small flocks clustered around small numbers of influencing agents; these flocks have converged in the sense that they are all afcing the same direction, but they remain disconnected from each other. An interesting challenge would be to develop algorithms to merge small flocks that start out with the same orientation, while maintaining flock composition. For this challenge, a successful algorithm for an influencing agent must change the direction of the flock without losing individual Reynolds-Vicsek agents on the edge of the flock.

Another direction would be to change the flocking dynamics and see how the various algorithms perform in that case. Our work has focused on a simplified version of the Reynolds flocking model and has only studied alignment dynamics. The original Reynolds model also includes avoidance and cohesion dynamics. These dynamics are important in real flocks, and in the future, the effects of including these dynamics should be explored.

A

Other Behaviors Explored

May include a bunch of other behaviors we played with here. Not sure yet.

References

[1] Forrest Briggs and Melissa O'Neill. 2008. Functional Genetic Programming and Exhaustive Program Search with Combinator Expressions. *Int. J. Know.-Based Intell. Eng. Syst.* 12, 1 (Jan. 2008), 47–68. http://dl.acm.org/citation.cfm?id=1375341.1375345

[2] Yufeng Chen, Hongqiang Wang, E. Farrell Helbling, Noah T. Jafferis, Raphael Zufferey, Aaron Ong, Kevin Ma, Nicholas Gravish, Pakpong Chirarattananon, Mirko Kovac, and Robert J. Wood. 2017. A biologically inspired, flapping-wing, hybrid aerial-aquatic microrobot. *Science Robotics* 2, 11 (2017).

[3] Iain D Couzin, Jens Krause, Nigel R Franks, and Simon A Levin. 2005. Effective leadership and decision-making in animal groups on the move. *Nature* 433 (Feb 2005).

[4] Leandro Nunes de Castro. 2007. *Evolutionary Computing*. Chapman & Hall/CRC, Chapter 3, 105–108.

[5] Marco Dorigo, Vito Trianni, Erol Şahin, Roderich Groß, Thomas H. Labella, Gianluca Baldassarre, Stefano Nolfi, Jean-Louis Deneubourg, Francesco Mondada, Dario Floreano, and Luca M. Gambardella. 2004. Evolving Self-Organizing Behaviors for a Swarm-Bot. *Auton. Robots* 17, 2-3 (Sept. 2004), 223–245. https://doi.org/10.1023/B:AURO.0000033973.24945.f3

[6] Katie Genter. 2017. *Fly with Me: Algorithms and Methods for Influencing a Flock*. Ph.D. Dissertation. The University of Texas at Austin.

[7] Katie Genter, Noa Agmon, and Peter Stone. 2013. Ad Hoc Teamwork for Leading a Flock. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*.

[8] Katie Genter, Noa Agmon, and Peter Stone. 2013. Improving Efficiency of Leading a Flock in Ad Hoc Teamwork Settings. In *AAMAS Autonomous Robots and Multirobot Systems (ARMS) Workshop*.

[9] Katie Genter and Peter Stone. 2014. Influencing a Flock via Ad Hoc Teamwork. In *Proceedings of the Ninth International Conference on Swarm Intelligence (ANTS 2014)*.

[10] Katie Genter and Peter Stone. 2016. Ad Hoc Teamwork Behaviors for Influencing a Flock. *Acta Polytechnica* 56, 1 (2016). https://doi.org/10.14311/APP.2016.56.0018

[11] Katie Genter and Peter Stone. 2016. Adding Influencing Agents to a Flock. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-16)*.

[12] Katie Genter, Shun Zhang, and Peter Stone. 2015. Determining Placements of Influencing Agents in a Flock. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS-15)*.

[13] José Halloy, Grégory Sempo, Gilles Caprari, Colette Rivault, Masoud Asadpour, Fabien Tâche, Imen Saïd, Virginie Durier, Stephane Canonge, J. M. Amé, Claire Detrain, Nikolaus Correll, A. Martinoli, Francesco Mondada, Roland Siegwart, and Jean-Louis Deneubourg. 2007. Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices. *Science* 318, 5853 (2007), 1155–1158.

[14] Jing Han, Ming Li, and Lei Guo. 2010. Soft Control on Collective Behavior of a Group of Autonomous Agents By a Shill Agent. 19 (07 2010).

[15] Ali Jadbabaie, Jie Lin, and A. Stephen Morse. 2003. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Automat. Control* 48, 6 (June 2003), 988–1001.

[16] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[17] Haili Liang, Yiping Yang, and Xiaofan Wang. 2013. Opinion dynamics in networks with heterogeneous confidence and influence. *Physica A* 392 (2013).

[18] Sean Luke. 1998. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo (Eds.). Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA, 214–222. http://www.cs.gmu.edu/~sean/papers/robocupgp98.pdf

[19] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. MASON: A Multi-Agent Simulation Environment. In *Simulation: Transactions of the society for Modeling and Simulation International*.

[20] Reza Olfati-Saber. 2006. Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Trans. Automat. Control* 51, 3 (March 2006), 401–420.

[21] Craig W. Reynolds. 1987. Flocks, Herds and Schools: A Distributed Behavioral Model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*.

[22] Lee W. Schruben and Dashi I. Singham. 2010. Simulating Multivariate Time Series Using Flocking. In *Proceedings of the 2010 Winter Simulation Conference*.

[23] Henry Segerman. 2010. The Sunflower Spiral and the Fibonacci Metric. (December 2010).

[24] Dashi Singham, Meredith Therkildsen, and Lee Schruben. 2011. Applications of Flocking Algorithms to Input Modeling for Agent Movement. In *Proceedings of the 2011 Winter Simulation Conference*.

[25] Housheng Su, Xiaofan Wang, and Zongli Lin. 2009. Flocking of Multi-Agents With a Virtual Leader. *IEEE Trans. Automat. Control* 54, 2 (Feb 2009), 293–307.

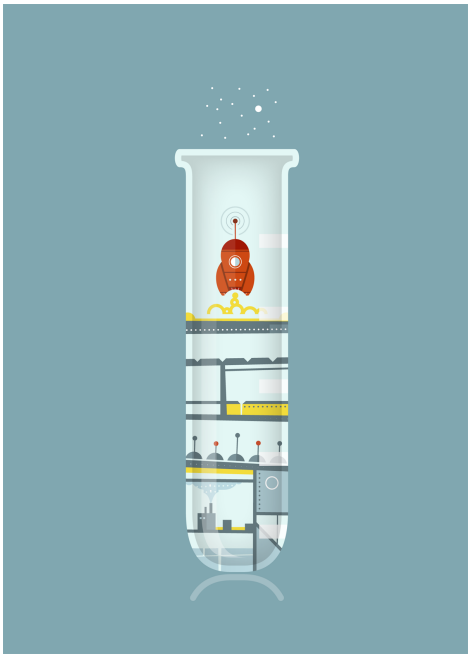
[26] David JT Sumpter. 2010. *Collective animal behavior*. Princeton University Press.

[27] David J.T. Sumpter, Jens Krause, Richard James, Iain D. Couzin, and Ashley J.W. Ward. 2008. Consensus Decision Making by Fish. *Current Biology* 18 (2008).

[28] Richard Vaughan, Neil Sumpter, Andy Frost, and Stephen Cameron. 1998. Robot Sheepdog Project achieves automatic flock control. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on the Simulation of Adaptive Behaviour*. MIT Press, 489–493.

[29] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. 1995. Novel Type of Phase Transition in a System of Self-Driven Particles. *Phys. Rev. Lett.* 75 (Aug 1995), 1226–1229. Issue 6.

[30] Wen Yang, Lang Cao, Xiaofan Wang, and Xiang Li. 2006. Consensus in a heterogeneous influence network. *Physical Review E* 74 (2006).



This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, *Science Experiment 02*, was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.