Acta Polytechnica **00**(0):1–8, 0000

PREPRINT 2015-09-24

To appear in *Acta Polytechnica Journal*, Jan/Feb, 2016.

AD HOC TEAMWORK BEHAVIORS FOR INFLUENCING A FLOCK

Katie Genter*, Peter Stone

Department of Computer Science, The University of Texas at Austin, Austin, TX, USA 78712

*corresponding author: katie@cs.utexas.edu

Abstract. Ad hoc teamwork refers to the challenge of designing agents that can influence the behavior of a team, without prior coordination with its teammates. This paper considers influencing a flock of simple robotic agents to adopt a desired behavior within the context of ad hoc teamwork. Specifically, we examine how the ad hoc agents should behave in order to orient a flock towards a target heading as quickly as possible when given knowledge of, but no direct control over, the behavior of the flock. We introduce three algorithms which the ad hoc agents can use to influence the flock, and we examine the relative importance of coordinating the ad hoc agents versus planning farther ahead when given fixed computational resources. We present detailed experimental results for each of these algorithms, concluding that in this setting, inter-agent coordination and deeper lookahead planning are no more beneficial than short-term lookahead planning.

Keywords: Ad Hoc Teamwork, Agent Cooperation, Coordination, Flocking.

1. Introduction

Consider a flock of migrating birds that is flying directly towards a dangerous area, such as an airport or a wind farm. It will be better for both the flock and the humans if the path of the migratory birds is altered slightly such that the flock can avoid the dangerous area but still reach their migratory point at approximately the same time.

The above scenario is a motivating example for our work in orienting a flock using ad hoc teamwork. We assume that each bird in the flock dynamically adjusts its heading based on that of its immediate neighbors. We assume further that we control one or more ad hoc agents — perhaps in the form of robotic birds or ultralight aircraft[1] — that are perceived by the rest of the flock as one of their own.

Flocking is an emergent behavior found in different species in nature including flocks of birds, schools of fish, and swarms of insects. In each of these cases, the animals follow a simple local behavior rule that results in a group behavior that appears well organized and stable. Research on flocking behavior has appeared in various disciplines such as physics [1], graphics [2], biology [3, 4], and distributed control theory [5–7]. In each of these disciplines, the research has focused mainly on characterizing the emergent behavior.

In this paper, we consider the problem of *leading* a team of flocking agents in an ad hoc teamwork setting. An ad hoc teamwork setting is one in which a teammate — which we call an *influencing agent* — must determine how to best achieve a team goal given a set of possibly suboptimal teammates. In this work, we are given a team of flocking agents following a known, well-defined rule characterizing their flocking behavior, and we wish to examine how the influencing agents should behave. Specifically, the main question addressed in this paper is: *how should influencing agents behave so as to orient the rest of the flock towards a target heading as quickly as possible?*

The remainder of this paper is organized as follows. Section 2 introduces our problem and necessary terminology for this paper. The main contribution of this paper is the 1-step lookahead algorithm for orienting a flock to travel in a particular direction. This algorithm is presented in Section 3, while variations of this algorithm are presented in Sections 4 and 5. We present the results of running experiments using these algorithms in the MASON simulator [8] in Section 6. Section 7 situates this research in the literature, and Section 8 concludes.

2. Problem Definition

In this work we use a simplified version of Reynolds' Boid algorithm for flocking [2]. We assume that each agent calculates its orientation for the next time step to be the average heading of its *neighbors*. Throughout this paper, an agent's *neighbors* are the agents located within some set radius of the agent. In order to calculate its orientation for the next time step, each agent computes the vector sum of the velocity vectors of each of its neighbors and adopts a scaled version of the resulting vector as its new orientation. An agent is not considered to be a neighbor of itself, so an agent's current heading is not considered when calculating its orientation for the next time step. Figure 1 shows an example of how an agent's new velocity vector is calculated. At each time step, each agent moves one step in the direction of its current vector

[1] www.operationmigration.org

and then calculates its new heading based on those of its neighbors, keeping a constant speed.

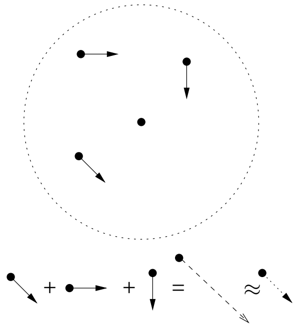


Figure 1. An example of how an agent's new velocity vector would be calculated. In this example, the black dot represents the agent in question, the solid arrows represent the velocity vectors of the agent's neighbors, and the dotted circle represents the area of the agent's neighborhood. The agent's new velocity vector is calculated as shown at the bottom of the figure — in this calculation, the three vectors are first summed and then scaled to maintain constant speed.

Over time, agents behaving as described above will gather into one or more groups, and these groups will each travel in some direction. However, in this work we add a small number of *influencing agents* to the flock. These influencing agents attempt to influence the flock to travel in a pre-defined direction — throughout this paper we refer to this desired direction as θ^*. Note that the challenge of designing influencing agent behaviors in a dynamic flocking system is difficult because the action space is continuous. Hence, in our work we make the simplifying assumption of only considering a limited number (*numAngles*) of discrete angle choices for each influencing agent.

2.1. Simulation Environment

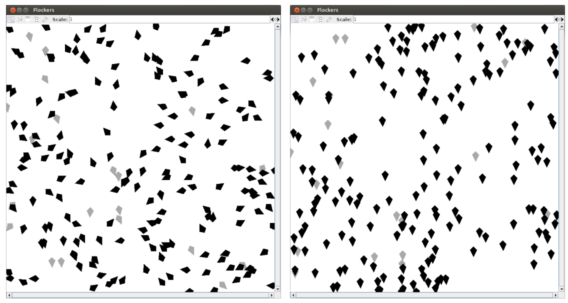
We situate our research on flocking using ad hoc teamwork within the MASON simulator, a concrete simulation environment [8]. A picture of the Flockers domain is shown in Figure 2. Each agent points and moves in the direction of its current velocity vector.

The MASON Flockers domain is toroidal, so agents that move off one edge of our domain reappear on the opposite edge moving in the same direction.

We conclude that the flock has converged to θ^* when every agent (that is not an influencing agent) is facing within 0.1 radians of θ^*. Other stopping criteria, such as when 90% of the agents are facing within 0.1 radians of θ^*, could also have been utilized.

3. 1-Step Lookahead Behavior

In this section we present Algorithm 1, a 1-step lookahead algorithm for determining the individual behavior of each influencing agent. 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.



(a) Start (b) Finish

Figure 2. Pictures of (a) the start of a trial and (b) the end of a trial in the MASON Flockers simulation environment. The grey agents are influencing agents while the black agents are other members of the flock.

The 1-step lookahead algorithm is a greedy, myopic approach 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. Note that if the algorithm only considered the current orientations of the neighbors (instead of the influences on these neighbors) when determining the next orientation for the influencing agent to adopt, it would only be estimating the state of each neighbor and hence the resulting orientation adopted by the influencing agent would not be 'best'.

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*
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 1. Variables used in Algorithm 1.

The variables used throughout Algorithm 1 are defined in Table 1. Two functions are used in Algorithm 1: *neighbor.vel* returns the velocity vector of neighbor and *neighbor.neighbors* returns a set containing the neighbors of neighbor.

Note that Algorithm 1 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 a parameter 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.

Algorithm 1 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
```

Conceptually, Algorithm 1 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 1 considers each of the *numAngles* discrete influencing agent orientation vectors. For each orientation 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 1 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 1 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 1 is $O(numAngles * numAgents^2)$.

Results regarding how Algorithm 1 performs in terms of the number of time steps needed for the flock to converge to θ^* can be found in Section 6.

4. 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 a 2-step lookahead behavior in Algorithm 2. This 2-step lookahead behavior 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 2 that were not used in Algorithm 1 are defined in Table 2. Like Algorithm 1, Algorithm 2 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.

Variable	Definition
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 2. Variables used in Algorithm 2 that were not used in Algorithm 1.

Algorithm 2 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
```

Conceptually, Algorithm 2 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 2) and (2) how the neighbors of the neighbors of each neighbor of the influencing agent are influenced if the influencing agent adopts a particular orientation at this time step (lines 19-25 in Algorithm 2), since they will influence the neighbors of each neighbor of the influencing agent on the next time step (lines 16-31 in Algorithm 2).

Algorithm 2 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 2 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 2 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 2 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 2 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 2 is $O(numAngles^2 * numAgents^3)$.

5. Coordinated Behavior

The influencing agent behaviors presented in Sections 3 and 4 were for individual influencing agents, where each influencing agent calculated its behavior independent of any other influencing agents. In this section, we consider whether influencing agents can exert more influence on the flock by working in a coordinated fashion. 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, in this work it is infeasible due to the complexity of such a calculation. Instead, 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.

We select the influencing agents to pair by first finding all pairs of influencing agents with one or more neighbors in common. Then we do a brute-force search and find every possible disjoint combination of these pairs. For each such combination, we calculate the sum of the number of shared neighbors across all the pairs and select the combination with the greatest sum of shared neighbors. 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 3 and coordinate with a partner influencing agent. If it is not part of a pair, it follows Algorithm 1 and performs a 1-step lookahead search for the best individual behavior.

Only one new variable and one new function are used in Algorithm 3 that are not used in Algorithm 1 or Algorithm 2. The variable is "nOrientsP", which is 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*. The function is *neighbors.get(x)*, which returns the xth element in the set *neighbors*.

Algorithm 3 is called on influencing agents that are part of a pair in *selectedPairs* at each time step. Algorithm 3 takes in the neighbors of the influencing agent and the neighbors of the partner of the influencing agent, and returns the orientation that, if adopted by the influencing agent, is guaranteed to influence the flock to face closer to θ^* than any of the other *numAngles* influencing agent orientations considered for both the influencing agent and its partner.

Conceptually, Algorithm 3 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 1 and Algorithm 3 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

Algorithm 3 bestOrient = 1StepCoordinated(neighOfIA, neighOfP)

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

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 3 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 the influencing agent adopts that orientation (lines 6-16). Then Algorithm 3 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 18-29). Finally, the algorithm selects the influencing agent orientation that results 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 30-34). Note that agents that are neighbors of both the influencing agent and its partner are only counted once (lines 28-29).

The complexity of Algorithm 3 can be calculated as follows. Line 3 executes *numAngles* times, line 4 executes *numAngles* times, line 6 executes at most *numAgents* times, line 8 executes at most *numAgents*, line 18 executes at most *numAgents* times, and line 20 executes at most *numAgents*. Hence, the complexity for Algorithm 3 is $O(numAngles^2 * numAgents^2)$.

Results for how Algorithm 3, as well as Algorithms 1 and 2, performed in our experiments can be found in the next section.

6. Experiments

In this section we describe our experiments testing the three influencing agent behaviors presented in Sections 3, 4, and 5 against some baseline methods described in this section. Our original hypothesis was that Algorithms 1, 2, and 3 would all perform significantly better than the baseline methods. We also believed that Algorithms 2 and 3 would perform better than Algorithm 1.

6.1. Baseline Ad Hoc 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 3, 4 and 5.

6.1.1. 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 [6]. 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^*.

6.1.2. 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 3 for an example calculation. In Figure 3, 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 influencing agent behavior was inspired by our previous work [9]. In this work, we showed how to optimally orient a stationary agent to a desired orientation using a set of stationary influencing agents. In particular, we presented an algorithm which the influencing agents could utilize to orient the agent to the desired orientation in the least number of steps possible. Our Offset Momentum influencing agent behavior implements this algorithm. However, this algorithm assumes that the agent is only influenced by influencing agents within its neighborhood. Hence, it is not optimal in our experimental setting because

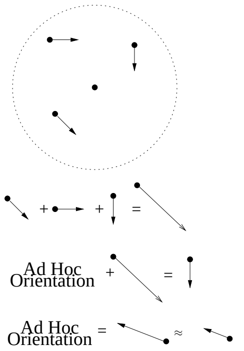


Figure 3. An example of how the Offset Momentum influencing agent behavior works. The influencing agent is the black dot, the circle represents the influencing agent's neighborhood, and the three arrows inside the circle represent the influencing agent's neighbors.

each agent being influenced by an influencing agent is usually also being influenced by other agents.

6.2. Experimental Setup

We utilize the MASON simulator [8] for our experiments in this paper. The MASON simulator was introduced in Section 2.1, but in this section we present the details of the environment that are important for completely understanding our experimental setup.

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.

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.

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 algorithm 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.

In all of our experiments, 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.

6.3. Experimental Results

Table 3 shows the number of time steps needed for the flock to converge to θ^* for the two baseline algorithms, the 1-step lookahead algorithm presented in Algorithm 1, the 2-step lookahead algorithm presented in Algorithm 2, and the coordinated algorithm presented in Algorithm 3 using the experimental setup described in Section 2.1.

Algorithm	**Time Steps**	**95% CI (±)**
Face Desired Orientation	34.82	3.85
Offset Momentum	36.70	4.63
1-Step Lookahead	26.02	3.10
2-Step Lookahead	25.94	3.16
Coordinated	25.76	3.15

Table 3. The number of time steps required for the flock to converge to θ^* using the experimental setup described in Section 2.1. CI stands for confidence interval.

The results shown in Table 3 clearly show 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.

6.3.1. 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 2.1. Altering the percentage of influencing agents in the flock clearly alters the amount of agents we can control, which affects the amount of influence we can exert over the flock. Hence, as can be seen in Figure 4, 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.

6.3.2. 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 2.1. Changing the flock size will alter the number of influencing agents, but not the ratio of influencing agents to

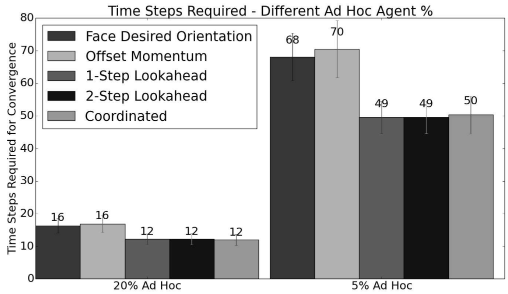


FIGURE 4. Results from experiments using the experimental setup described in Section 2.1, except that we varied the percentage of influencing agents in the flock. The values in the table are averaged over 50 trials and the error bars represent the 95% confidence interval.

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 5.

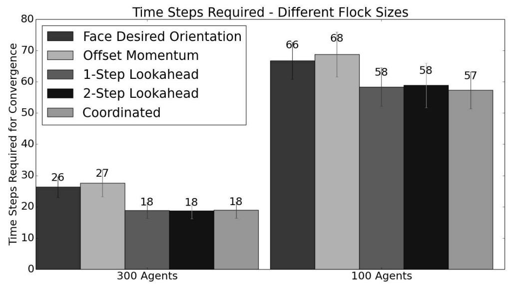


FIGURE 5. Results from experiments using the experimental setup described in Section 2.1, except that we varied number of agents in the flock. The values in the table are averaged over 50 trials and the error bars represent the 95% confidence interval.

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.

6.4. DISCUSSION

Our hypothesis was that Algorithms 1, 2, and 3 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.

Our 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 paper, we found that it is best for influencing agents to perform the 1-step lookahead behavior presented in Algorithm 1. This behavior is more computationally efficient than the other two algorithms presented, and performed significantly better than the baseline methods in most cases.

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 were often behaved similarly, and hence did not exert significantly different types of influence.

There are, of course, cases in which each of the lookahead and coordinated behaviors perform noticeably better than the others. For example, when the flock size is decreased to 100, the 2-step lookahead only takes 44 time steps to converge to θ^* when a particular random seed (93) is used in the simulator, but the 1-step lookahead takes 67 steps and the coordinated approach takes 61 steps.

7. RELATED WORK

Although there has been a significant amount of work in the field of multiagent teamwork, there has been relatively little work towards getting agents to collaborate with teammates that can not be explicitly controlled. Most prior multiagent teamwork research requires explicit coordination protocols or communication protocols (e.g. SharedPlans, STEAM, and GPGP) [10–12]. However, in our work we do not assume that any protocol is known by all agents.

Han, Li and Guo studied how one agent can influence the direction in which an entire flock of agents is moving [5]. Similarly to our work, in their work each agent follows a simple control rule based on its neighbors. However, unlike our work, they only consider one influencing agent with unlimited, non-constant velocity. This allows their influencing agent to move to any position in the environment within one time step, which we believe is unrealistic.

As we mention in Section 2, Reynolds introduced the original flocking model [2]. However, his work focused on creating graphical models that looked and behaved like real flocks, and hence he did not address adding controllable agents to the flock like we do.

Vicsek et al.considered just the alignment aspect (also called flock centering) of Reynolds' model [1]. Hence, like in our work, they 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, they were only concerned with simulating flock behavior and not with adding controllable agents to the flock.

Jadbabaie, Lin, and Morse build on Vicsek *et al.*'s work [6]. They use a simpler direction update than Vicsek et al.and 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. However, they are only concerned with getting the flock to converge eventually, whereas we attempt to do so as quickly as possible. Su, Wang, and Lin also present work that is concerned with using a controllable agent to make the flock converge eventually [7].

8. Conclusion

In this work, we set out to determine how influencing agents should behave in order to orient a flock towards a target heading as quickly as possible. Our work is situated in a limited ad hoc teamwork domain, so although we have knowledge of the behavior of the flock, we are only able to influence them indirectly via the behavior of the influencing agents within the flock. This paper introduces three algorithms that the influencing agents can use to influence the flock — a greedy lookahead behavior, a deeper lookahead behavior, and a coordinated behavior. We ran extensive experiments using these algorithms in a simulated flocking domain, where we observed that in such a setting, a greedy lookahead behavior is an effective behavior for the influencing agents to adopt.

Although we begin to consider coordinated algorithms in this work, there is room for more extensive coordination as well as different types of coordination. Additionally, as this work focused on a limited version of Reynolds' flocking model, a promising direction for future work is to extend the algorithms presented in this work to Reynolds' complete flocking model. Finally, it would be interesting to empirically consider the effect of influencing agent placement.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-1330072, CNS-1305287), ONR (21C184-01), AFRL (FA8750-14-1-0070), AFOSR (FA9550-14-1-0087), and Yujin Robot.

References

[1] T. Vicsek, A. Czirok, E. Ben-Jacob, et al. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters* **75**(6), 1995. DOI:10.1103/PhysRevLett.75.1226.

[2] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH* **21**:25–34, 1987. DOI:10.1145/37401.37406.

[3] W. Bialeka, A. Cavagnab, I. Giardinab, et al. Statistical mechanics for natural flocks of birds. *Proceedings of the National Academy of Sciences* **109**(11), 2012. DOI:10.1073/pnas.1118633109.

[4] H. H. Charlotte K. Hemelrijk. Some causes of the variable shape of flocks of birds. *PLoS ONE* **6**(8), 2011. DOI:10.1371/journal.pone.0022479.

[5] J. Han, M. Li, 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. DOI:10.1007/s11424-006-0054-z.

[6] A. Jadbabaie, J. Lin, A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control* **48**(6):988–1001, 2003. DOI:10.1109/TAC.2003.812781.

[7] H. Su, X. Wang, Z. Lin. Flocking of multi-agents with a virtual leader. *IEEE Transactions on Automatic Control* **54**(2):293–307, 2009. DOI:10.1109/TAC.2008.2010897.

[8] S. Luke, C. Cioffi-Revilla, L. Panait, et al. Mason: A multi-agent simulation environment. *Simulation: Transactions of the Society for Modeling and Simulation International* **81**(7):517–527, 2005. DOI:10.1177/0037549705058073.

[9] K. Genter, N. Agmon, P. Stone. Ad hoc teamwork for leading a flock. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*. 2013.

[10] B. J. Grosz, S. Kraus. Collaborative plans for complex group action. *Artifical Intelligence Journal* **86**(2):269–357, 1996. DOI:10.1016/0004-3702(95)00103-4.

[11] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research* **7**:83–124, 1997.

[12] K. S. Decker, V. R. Lesser. Readings in agents. chap. Designing a family of coordination algorithms, pp. 450–457. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.