In *AAMAS Autonomous Robots and Multirobot Systems Workshop* (ARMS13), Saint Paul, MN, USA, May 2013.

Improving Efficiency of Leading a Flock in Ad Hoc Teamwork Settings

Katie Genter[1], Noa Agmon[2], and Peter Stone[1]

[1] University of Texas at Austin, Austin TX 78701, USA
{katie, pstone}@cs.utexas.edu
[2] Bar Ilan University, Ramat Gan, 52900, Israel
agmon@cs.biu.ac.il

Abstract. Designing robots that can influence their teammates to behave in a particular manner is desirable — especially if the robots are able to do this without prior coordination or explicit communication. In this paper, we examine an aspect of the problem of leading teammates in an ad hoc teamwork setting. Specifically, we consider how to best utilize ad hoc agents to lead a flock to a desired orientation. Specifically, we extend upon recent work [4] by presenting a more efficient search methodology for determining how the ad hoc agents should orient themselves to optimally influence a flock.

1 Introduction

Robots are being designed to work in various cooperative domains by an ever increasing number of parties. Hence, it is important that these robots be capable of reasoning about *ad hoc* teamwork [10]. Robots that can reason about ad hoc teamwork can cooperate within a team without needing explicit communication and without needing to previously coordinate behaviors with their teammates.

Consider a platoon of robots attempting to travel from the parking lot at a local park to a bridge that needs to be repaired. The terrain between the parking lot and the bridge is varied, and the straight line path between the two locations will be difficult for the robots to traverse. Now consider that most of the robots that will be traveling from the parking lot to the bridge have a very simple method of planning a path to travel: they generally head directly towards their goal location but will adopt the orientation of any teammates within close proximity. If allowed to traverse from the parking lot to the bridge on their own, these simple robots will need to traverse steep inclines and very dense brush. However, consider if a few more sophisticated robots were included in the platoon that could observe the path planning method used by the simple robots and had some previous knowledge about the terrain of the park. These more sophisticated robots could then influence the simpler robots to alter their path (since the simpler robots adopt the orientation of any teammates within close proximity) such that the worst of the steep inclines and dense brush would be avoided. This would allow the entire platoon to reach the bridge in less time and with less damage to the robots. Note that the more sophisticated robots can not just plan optimally for themselves, as doing so may not influence the simpler robots to take a path that avoids the worst of the steep inclines and dense bush.

In this example, the more sophisticated robots influenced — or *led* — the simpler robots to adopt a better path from the parking lot to the bridge. One aspect of ad hoc teamwork involves this idea of *leading* teammates to perform desired actions or adopt particular behaviors. In particular, this paper builds upon our recent work concentrated on leading flocking agents [4].

Flocking is a behavior found in various 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 stable, well defined group behavior. Research on flocking behavior can be found in various disciplines such as physics [13], graphics [9], biology [1, 2], and distributed control theory [6, 7, 11]. The main focus of each of these research directions is to characterize the resulting group behavior. In this paper we consider the problem of leading a team of flocking agents in an ad hoc teamwork setting. Specifically, we are given a team of flocking agents following a known, well-defined rule characterizing the flock's behavior and we wish to examine to what extent it is possible to influence the flock.

The ad hoc teamwork perspective of this problem is highlighted by two facts. Firstly, we are unable to explicitly control the behavior of the flocking agents, thus we can only attempt to influence them implicitly using the behavior of the ad hoc agents. The flock does not know that the ad hoc agents are different than any other members of the flock, and hence each ad hoc agent is able to influence the flock the same as any other member of the flock. Secondly, all agents — both flocking and ad hoc — act as one team, and their only desire is to optimize team utility. The ad hoc agents cannot communicate with the flocking agents, but they can coordinate actions among themselves.

This paper is based upon work presented at AAMAS'13 [4], and extends upon the work presented in that paper. In the AAMAS'13 paper we presented a specification for the flocking problem as a new scenario for studying ad hoc teamwork and an initial theoretical and empirical analysis. Specifically, we set bounds on the extent of influence the ad hoc agents can have on the team when all the agents have zero velocity, and we provided an empirical evaluation of the suggested solution using our custom-designed simulator FlockSim.

One of the main contributions of this paper is a new search methodology for determining how the ad hoc agents should orient themselves to optimally influence a flock. This search methodology is much more efficient than the one presented in our AAMAS'13 work. Another major contribution of this paper is a discussion regarding how the ad hoc agents can improve their leading behavior when they are able to move with non-zero velocity. A review of the problem definition and important concepts from our AAMAS'13 work is presented in Section 2. The improved search methodology for determining how the ad hoc agents should orient themselves to optimally influence a flock is presented in Section 3, while the discussion of improvements in the ad hoc agents' leading behavior when they are able to move with non-zero velocity is given in Section 4. Section 5 situates this research in the literature, and Section 6 concludes.

2 Problem Definition

In recent work [4], we introduced a flocking model that was initially inspired by Vicsek *et al.* [13]. For clarity, we summarize the important points that will be utilized in this paper in this section.

In our flocking model, n agents which are visually indistinguishable inhabit some environment where each agent a_i moves with some velocity v_i. At each time step t, each agent a_i has a position $p_i(t) = (x_i(t), y_i(t))$ in the environment and an orientation $\theta_i(t)$. Each agent's position $p_i(t)$ at time t is updated after its orientation is updated, such that

$$x_i(t) = x_i(t-1) + v_i \cos(\theta_i(t))$$

$$y_i(t) = y_i(t-1) - v_i \sin(\theta_i(t))$$

We let $N_i(t)$ be the set of $n_i(t) \leq n$ agents (including agent a_i) at time t which are visible to agent a_i. An agent is *visible* to agent a_i if its position is located within a *visibility cone* of angle α centered on orientation $\theta_i(t)$ and extending from agent a_i for an unlimited distance (see Figure 1 for an example). We say that angle α defines the visibility cone for each agent. Under our flocking model, the global orientation of agent a_i at time step $t+1$, $\theta_i(t+1)$, is set to be the average orientation of all agents in $N_i(t)$ (including itself) at time t.

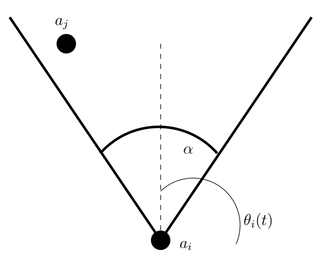

Fig. 1: Angle α defines the visibility cone for agent a_i. Agent a_j is visible to agent a_i.

A *border agent* is an ad hoc agent that is located within the visibility cone of the flocking agents, on the edge of the visibility cone that is farthest to the target. A *border influence orientation* is a flocking agent orientation at which an ad hoc agent is a border agent. See Figure 2 for an example with a border agent.

The n agents that comprise the flock consist of k *ad hoc* agents and m *flocking* agents, where $k+m = n$. Note that the k *ad hoc* agents and m *flocking* agents are visually indistinguishable. The ad hoc agents $\{a_0, \ldots, a_{k-1}\}$ are agents whose

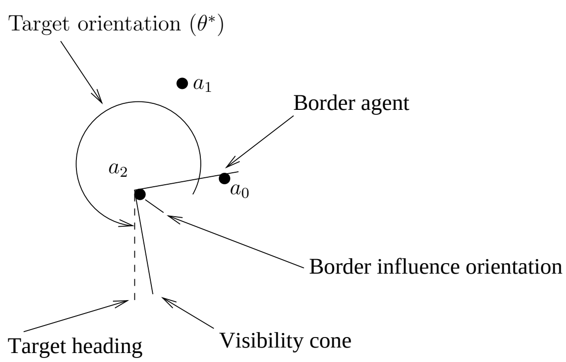

Fig. 2: An example of a border agent (a_0) and the resulting border influence orientation of the flocking agent (a_2).

behavior we can control directly, while the flocking agents $\{a_k, \ldots, a_{n-1}\}$ are agents that we cannot directly control. In this paper, we make the simplification that although we can have many flocking agents, they all must have the same position $p(t)$ in the environment.

An x-step plan specifies the orientations that each ad hoc agent $\{a_0, a_1, \ldots a_{k-1}\}$ will align to at each time step when given exactly x time steps in which to act.

3 Backward Search Approach

In recent work [4], we present a forward search approach for determining how the ad hoc agents should orient themselves to optimally influence the flocking agents. The forward search works as follows. Beginning at the initial flocking orientation, we consider each possible border influence orientation. If the border influence orientation is reachable from the initial flocking orientation, then we consider each possible border influence orientation from this orientation. If the border influence orientation is not reachable from the initial flocking orientation, then we turn to the farthest reachable orientation and then determine if the border influence orientation is now reachable (and repeat this process until the border influence orientation is reachable). We repeat this entire process until the target is within reach, and we select the plan that reaches the target in the fewest number of steps.

This forward search approach works in a very intuitive manner, but it requires checking 2^k possible combinations of the number of ad hoc agents influencing the flocking agents at each time step. For example, with three ad hoc agents, the following eight combinations of targets must be checked at each time step: $[a_0, a_1, a_2]$, $[a_0, a_1]$, $[a_0, a_2]$, $[a_0]$, $[a_1, a_2]$, $[a_1]$, $[a_2]$, $[]$. Hence, in this section we present a more efficient backward search approach for determining how the ad hoc agents should orient themselves to optimally influence the flocking agents.

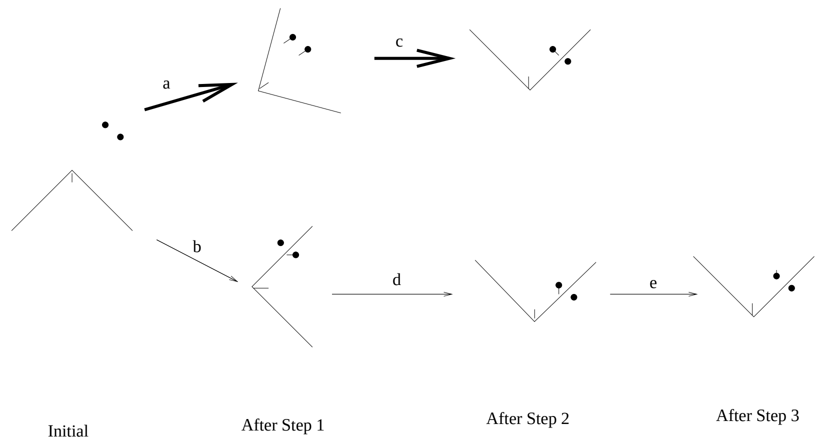

Fig. 3: Example showing how the backward search works. Each black dot represents an ad hoc agent, where the ad hoc agents' orientations (for those within the flocking agent's visibility cone) are represented by lines, while each angle represents the visibility cone of a flocking agent. Each arrow represents a possible step in the backward search — the letters labelling each arrow are referenced in the paragraph that explains the backward search. Each search terminates in the initial configuration. Note that the best flocking sequence (shown by larger arrows) takes 2 steps.

To assist in understanding how the backward search approach works, consider the example provided in Figure 3. Each possible step in Figure 3 is labeled with a letter, which we use to refer to the steps as we explain how the backward search approach works. We start by considering the farthest orientations from the target orientation at which the flocking agents can be and still reach the target orientation in one step when influenced by various numbers of ad hoc agents (a and b). We add each of the orientations from which the target orientation is reachable to a separate list. Then, while none of the orientations in the front of the lists are reachable from the initial orientation, we consider the farthest orientations from the orientation at the front of each list that the flocking agent can be and still reach the orientation at the front of the list in one step when influenced by various amounts of ad hoc agents (d). We continue this process until an orientation at the front of a list is reachable from the flocking agents' initial orientation (c and e). We then know the orientations in this list are the desired orientations for the flocking agents at each time step. A list of necessary ad hoc agent orientations is also built during this search.

Algorithm 1 uses such a backward search approach to calculate and return the number of steps needed to reach θ^* and the necessary orientations for each of the ad hoc agents for each of these steps. Throughout the algorithm, element.get(x) returns the 0-indexed x item in element (where element is a list object), element.add(y) adds item y to the end of element, element.add(x,y) adds item y at index x of the element, element.add(empty, x, y) adds a list containing x at index

0 and y at index 1 to element, element.remove(x) removes the item at index x of element, element.size() returns the number of items contained in element, and last is the index of the last item of the element. The variables used throughout Algorithm 1 are defined in Table 1. Remember that although the $k_i(t)$ ad hoc agents are located at many arbitrary locations, the $m_i(t)$ flocking agents are located at a single position p_i and begin with identical orientations.

Variable	**Definition**
adHocOrient	the orientation the ad hoc agents must adopt at this time step in order for the flocking agents to reach *target* from *current*
bestAHPlan	the ad hoc agent plan that uses the least number of time steps to reach θ^*
bestFSeq	the flocking sequence of orientations that uses the least number of time steps to reach θ^*
ccw	whether the flocking agents are rotating counter-clockwise
current	the orientation the flocking agents are currently oriented towards
currentAHPlan	the plan containing the orientations for each ad hoc agent at each time step so far
currentFSeq	the sequence of orientations for the flocking agents at each time step so far
initFOrient	the initial orientation of the flocking agents
maxSteps	the maximum number of steps a plan can be
numAH	the number of ad hoc agents within the flocking agents' visibility cone
m	the number of flocking agents
target	the orientation the flocking agents should be oriented towards on the next time step
k	the number of ad hoc agents

Table 1: Variables used in Algorithm 1.

In the worst case, line 4 of Algorithm 1 executes maxSteps times, line 5 executes k^2 times, line 8 executes k times, line 16 executes k times, and line 22 executes k times. Hence, lines 5-25 execute at most maxSteps $*\,(2k^2)$ times. To compare, the main part of the forward search presented in [4] executed at most $(2^k)(k+1)$(maxSteps) times.

The backward search presented here is much more efficient than the forward search presented in [4]. We ran tests to record the time required for both a forward search and a backward search on 100 runs, where the same randomization seed was used for both searches. For these experiments, $v = 0$, $\alpha = 90°$, $\theta^* = 270°$, the initial flocking orientation is 90°, and the ad hoc agents and flocking agents are placed randomly in a 950 by 500 environment. When run with teams composed of one to four ad hoc agents and one to two flocking agents on a Dell Precision-T3500 desktop computer, an optimal plan is found in 0.013 seconds on average by the forward search and in 0.005 seconds on average by the backward search. Hence, the backward search presented in this paper is 2.6 times faster than the forward search presented in [4] for the experimental setup just described.

Although the backward search presented here is more efficient, there are indeed benefits to utilizing a forward search. One such benefit becomes clear when we consider ad hoc agents with non-zero velocities. In this case, if a backward

Algorithm 1 plan, steps = calcPlan()

```
target ← θ*
currentFSeq ← empty list
currentAHPlan ← empty list
while no plan ∈ currentFSeq is reachable by initFOrient ∨ larger than maxSteps do
  for j = 0 to currentFSeq.size() do
    if currentFSeq.size() > 0 then
      target ← currentFSeq.get(j).get(0)
    for i = 1 to k do
      if ccw then
        current ← target − numAHπ/(numAH+m)
      else
        current ← target + numAHπ/(numAH+m)
      if i == numAH ∧ target == θ* then
        currentFSeq.add(empty, current, target)
        currentAHPlan.add(empty list of lists)
        for each ad hoc agent x in the flocking agents' visibility cone when facing current do
          currentAHPlan.get(last).get(x).add(adHocOrient)
      else if i == numAH then
        currentFSeq.add(currentFSeq.get(j))
        currentFSeq.get(last).add(0, current)
        currentAHPlan.add(currentAHPlan.get(j))
        for each ad hoc agent x in the flocking agents' visibility cone when facing current do
          currentAHPlan.get(last).get(x).add(adHocOrient)
    currentFSeq.remove(j)
    currentAHPlan.remove(j)
bestFSeq ← currentFSeq.get(k), where currentFSeq.get(k).get(0) is reachable by initFOrient
bestAHPlan ← currentAHPlan.get(k)
return bestAHPlan
```

search is used, we are unable to determine exactly where each ad hoc agent will be in the final steps of the plan, because the earlier steps of the plan will not have been determined when we are determining the later steps. We consider one solution to this problem later in this paper, but utilizing a forward search makes planning for ad hoc agents with non-zero velocities easier and more intuitive. Nonetheless, the efficiency gained by using a backward search such as described here is necessary as we continue to scale our work up to utilize more ad hoc agents.

Theorem 1. *Given θ^* and assuming the $m_i(t)$ flocking agents are influenced only by the $k_i(t)$ ad hoc agents and the $m_i(t)$ flocking agents at time t, then if θ^* is reachable, the ad hoc agents are guaranteed to lead the flocking agents to θ^* in the least number of time steps possible when the ad hoc agents determine their plan based on Algorithm 1.*

Proof Sketch: Line 4 ensures that the search will stop once one or more plans in currentFSeq are reachable by initFOrient.

Throughout the first iteration of line 4, the target will equal θ^*. If it is possible to reach θ^* from initFOrient in some number of steps less than maxSteps, then at least one orientation will be found on the first iteration of line 4 from which θ^* can be reached.

During subsequent iterations of line 4, target will be set to be some orientation that is closer to initFOrient than θ^*. During each iteration, if it is possible to reach θ^* from initFOrient in some number of steps less than maxSteps, then at least one orientation will be found on the each iteration of line 4 from which target can be reached. Hence, each iteration of line 4 will result in at least one flocking sequence being improved such that the orientation at the front of the sequence is closer to initFOrient than the value that was previously at the front of the flocking sequence (and that is now the second orientation in the flocking sequence).

If it is possible to reach θ^* from initFOrient in some number of steps less than maxSteps, one flocking sequence must eventually be improved such that the element at the front of the flocking sequence is reachable from initFOrient. We know this flocking sequence is optimal length, since if an optimal flocking sequence of lesser length existed, it would have been discovered on a previous iteration of line 4. □

4 Utilizing Ad Hoc Agents with Non-zero Velocity

In the previous section we showed how the backward search approach utilized in Algorithm 1 can be used to calculate the necessary orientations for each ad hoc agent to orient the flocking agent towards a target orientation θ^*. However, in the previous section we only considered ad hoc agents with zero velocity — in other words, stationary ad hoc agents. In this section we consider how using ad hoc agents with non-zero velocity can improve the ad hoc agents' ability to influence the flocking agents to turn towards θ^*.

For some configurations of ad hoc agents and flocking agents, ad hoc agents with non-zero velocity can influence the flocking agents to reach their target orientation in fewer time steps than ad hoc agents with zero velocity. We state this in Theorem 2.

Theorem 2. *Non-stationary ad hoc agents can have additional influence on the flocking agents compared to stationary ad hoc agents; specifically, the added velocity can influence the flocking agents to reach θ^* quicker.*

Proof. We prove this via an existence proof - consider the example presented in Figure 4. The ad hoc agents' positions are represented by black dots, the ad hoc agents' orientations (for those within the flocking agent's visibility cone) are represented by lines, the flocking agent is represented by the vertex of its visibility cone, and any movement to occur before the next step is represented by arrows.

Three steps are required for the flocking agent to reach θ^* when stationary ad hoc agents are utilized. However, when the ad hoc agents have non-zero velocities, an additional ad hoc agent is able to enter the flocking agent's visibility cone for the second time step. The two ad hoc agents are then able to influence the flocking agent to turn to θ^* on the second time step in the non-stationary ad hoc agents case. Hence, in this case the added velocity did influence the flocking agent to reach θ^* quicker. □

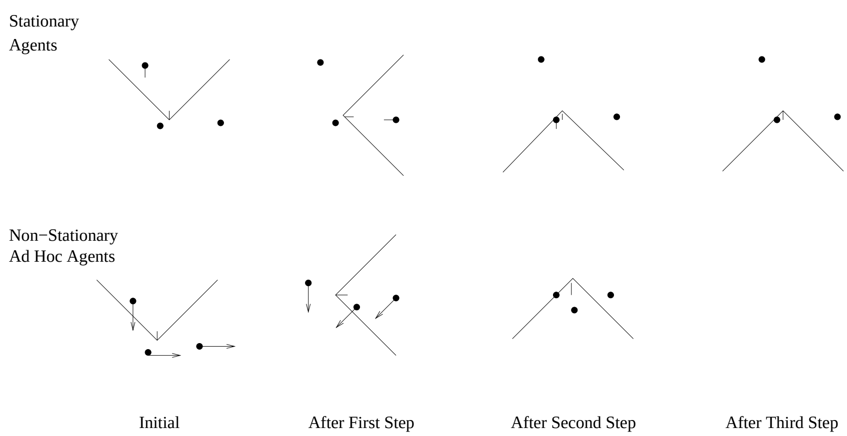

Fig. 4: The existence proof example presented in the proof for Theorem 2. In this example, the initial flocking orientation is 90° and $\theta^* = 270°$.

In some cases, ad hoc agents with non-zero velocity actually have less influence over the flocking agents than ad hoc agents with zero velocity. This statement may seem counter-intuitive. However, an example of such a case is provided in Figure 5 as part of the proof sketch for Theorem 3.

Theorem 3. *Non-stationary ad hoc agents can influence the flocking agents to reach θ^* slower than stationary ad hoc agents.*

Proof Sketch: We argue this via an existence proof - consider the example presented in Figure 5. The ad hoc agents' positions are represented by black dots, the ad hoc agents' orientations (for those within the flocking agent's visibility cone) are represented by lines, the flocking agent is represented by the vertex of its visibility cone, and any movement to occur before the next step is represented by arrows.

Two steps are required for the flocking agent to reach θ^* when stationary ad hoc agents are utilized. When the ad hoc agents have non-zero velocities, however, one of the ad hoc agents that is within the flocking agent's visibility cone after the first step in the stationary ad hoc agents case is not within the flocking agent's visibility cone in the non-stationary ad hoc agents case. This is because its velocity in the first step carried it outside the visibility cone for the second step. This led to three steps being required to turn the flocking agent to θ^* in the non-stationary ad hoc agents case. Hence, in this case the added velocity caused the optimal plan to influence the flocking agent to reach θ^* slower. $\square$

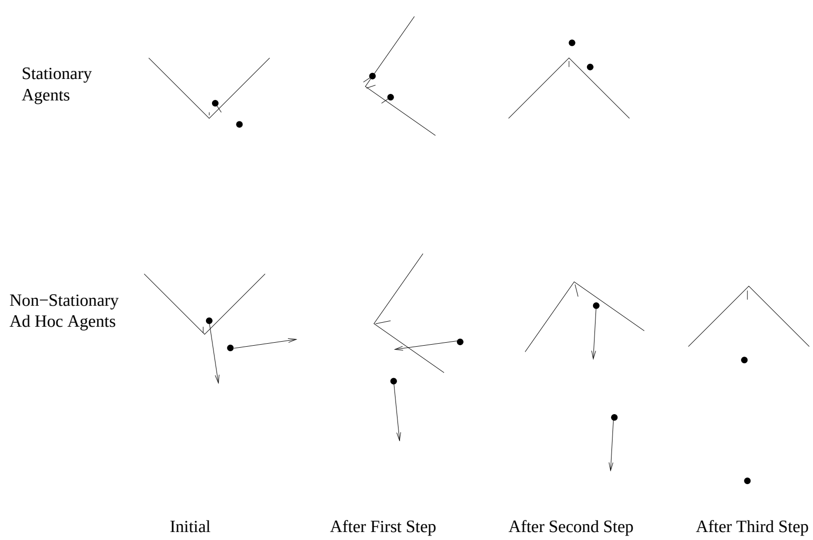

Fig. 5: The existence proof example presented in the proof for Theorem 3. In this example, the initial flocking orientation is $90°$ and $\theta^* = 270°$.

Of course, if non-stationary ad hoc agents are able to choose their velocity, they can do no worse than stationary ad hoc agents because they could choose to adopt a velocity of zero and hence perform the same as the stationary ad hoc agents.

4.1 Altering Ad Hoc Agent Behavior

The backward search approach presented in Algorithm 1 finds optimal plans for ad hoc agents with zero velocities. In this subsection, we present a methodology to alter ad hoc agent plans output from Algorithm 1 such that they can be used by ad hoc agents with non-zero velocities.

Ad hoc agents with non-zero velocities may move such that they are not within the flocking agents' visibility cone at time steps when they would have been had they been stationary. Likewise, ad hoc agents with non-zero velocities may move such that they are within the flocking agents' visibility cone at time steps when they would not have been had they been stationary. To make the output plan from Algorithm 1 usable for ad hoc agents with non-zero velocities, we run an alteration method over the output plan. This alteration method keeps the same desired sequence of orientations for the flocking agents, but recalculates the ad hoc agent orientations as needed to account for the possibility of more or less ad hoc agents being within the flocking agents' visibility cone during some time steps. Note that in some cases, the alteration method may not be able to alter the ad hoc agent behavior such that the flocking agents will still be influenced as they would have been had the ad hoc agents been stationary.

4.2 Replanning Ad Hoc Agent Behavior

In some cases, the alteration method discussed above results in ad hoc agents with non-zero velocities influencing the flocking agents to their target orientation quicker than ad hoc agents with zero velocities. However, now that we are dealing with ad hoc agents with non-zero velocities, we can purposely use their movement to influence the flocking agents to their target orientation quicker. In this section, we present two plan repair methods that can improve upon the ad hoc plans found by Algorithm 1 and altered using the method described above.

The ad hoc agent orientations (and hence direction of movement) are set by Algorithm 1 (and altered as described above) at each time step such that the ad hoc agents within the flocking agents' visibility cone guide the flocking agent to the desired orientation at the next time step. The ad hoc agents that are outside of the flocking agents' visibility cone, however, do not influence the flocking agents' orientation at the next time step. Hence, how we set their orientations has no influence on the flocking agents, since these ad hoc agents are not visible to the flocking agents. As such, we can set the orientations of these ad hoc agents such that they move in a particular direction or towards a particular point and potentially have more influence over the flocking agents at a future step.

The first plan repair method calls for at least one ad hoc agent to be in the flocking agents' visibility cone during at least one time step when it was not under the stationary plan. Having an additional ad hoc agent inside the visibility cone of the flocking agents allows the ad hoc agents to more strongly influence the flocking agents towards the target orientation θ^*. The second plan repair method calls for at least one ad hoc agent that was a border agent at a particular time step to move such that the border influence orientation of the flocking agent can be closer to θ^*. Allowing the flocking agents to be oriented closer to θ^* could allow the flocking agent to be influenced (in this step or a future step) to orient to θ^* in one fewer step than before.

It is possible that not all plans returned by Algorithm 1 will be able to be repaired to an optimal plan using the plan repair methods presented above. If our goal is to minimize the number of steps required to orient the flocking agents to θ^*, we may need to consider other plans that were found by Algorithm 1 but pruned out. Often Algorithm 1 finds multiple minimal-step plans, but it picks only one to return. We believe it is sufficient to consider repairing all the minimal-step plans found by Algorithm 1 — and that no better plan could be found by considering repairing any longer plans found by Algorithm 1. This notion is stated as Conjecture 1.

Conjecture 1. Running the two plan repair methods described above on all the minimal size plans returned by Algorithm 1 will obtain an optimal plan.

We believe Conjecture 1 to be true because any $X + 1$-step plans must have an unnecessary step.

If we assume Conjecture 1 is true, then it is natural to consider whether *all* of the minimal size plans returned by Algorithm 1 must be considered or whether it is sufficient to consider just *one* of the minimal size plans. Additionally, if it is sufficient to consider just one minimal size plan — can any minimal size plan be repaired to an optimal plan or can only some minimal size plans be repaired to an optimal plan? These are all currently open questions.

5 Related Work

Although there has been much work in the field of multiagent teamwork, there has been relatively little work towards getting agents to collaborate towards a common goal without pre-coordination. This paper contributes towards answering the ad hoc teamwork challenge [10]. Most prior multiagent teamwork research requires explicit coordination protocols or communication protocols (e.g. SharedPlans, STEAM, and GPGP) [5, 12, 3]. However, our work is different in that we do not assume that any protocol is known by all agents.

Han, Li and Guo study how one agent can influence the direction in which an entire flock of agents is moving [6]. 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 ad hoc agent with unlimited, non-constant velocity. This allows their ad hoc agent to move to any position in the environment within one time step, which is rather unrealistic.

Reynolds introduced the original flocking model when he presented three flocking behaviors — collision avoidance, velocity matching, and flock centering [9]. His work was focused on creating graphical models that looked and behaved like real flocks, and hence did not consider adding controllable agents to the flock like we do.

Vicsek *et al.* considered just the flock centering aspect of Reynolds' model [13]. Hence, 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 paper [7]. 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 would like 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 [11].

Jones *et al.* perform an empirical study of dynamically formed teams of heterogeneous robots in a multirobot treasure hunt domain [8]. They assume that all of the robots know they are working as a team and that all of the robots can communicate with one another, whereas in our work we do not assume that the teammates realize they are working on a team with the ad hoc agents.

6 Conclusions

In this paper, we build on our recent work [4] by continuing to consider the problem of leading a flock to a desired orientation using ad hoc agents. This paper's main contributions are (1) a more efficient search methodology for determining how the ad hoc agents should orient themselves to optimally influence a flock and (2) a preliminary look into how the ad hoc agents can improve their leading behavior when they are able to move with non-zero velocity.

Although we did consider the non-stationary ad hoc agent case in this work, there is much more to be studied with regard to this case. As noted in this paper, if we have a minimal ad hoc agent plan for stationary agents, it remains to be seen whether only this minimal plan must be repaired, whether all minimal plans originally found must be repaired, or even whether all plans originally found must be repaired, in order to obtain an optimal plan for non-stationary ad hoc agents. Additionally, the non-stationary ad hoc agent case is just an initial step towards solving the general case of non-stationary ad hoc and flocking agents. As such, we do plan to continue extending the work presented here towards this general case.

References

1. W. Bialeka, A. Cavagnab, I. Giardinab, T. Morad, E. Silvestrib, M. Vialeb, and A. Walczak. Statistical mechanics for natural flocks of birds. *Proceedings of the National Academy of Sciences*, 109(11), 2012.
2. H. H. Charlotte K. Hemelrijk. Some causes of the variable shape of flocks of birds. *PLoS ONE*, 6(8), 2011.

3. K. S. Decker and V. R. Lesser. Readings in agents. chapter Designing a family of coordination algorithms, pages 450–457. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
4. K. Genter, N. Agmon, and P. Stone. Ad hoc teamwork for leading a flock. In *AAMAS'13*, May 2013.
5. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *AIJ*, 86(2):269 – 357, 1996.
6. J. Han, M. Li, and L. Guo. Soft control on collective behavior of a group of autonomous agents by a shill agent. *Systems Science and Complexity*, 19:54–62, 2006.
7. A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988 – 1001, June 2003.
8. E. Jones, B. Browning, M. B. Dias, B. Argall, M. M. Veloso, and A. T. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *ICRA'06*, pages 570 – 575, 2006.
9. C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH*, 21:25–34, August 1987.
10. P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI'10*, 2010.
11. H. Su, X. Wang, and Z. Lin. Flocking of multi-agents with a virtual leader. *IEEE Transactions on Automatic Control*, 54(2):293–307, Feb 2009.
12. M. Tambe. Towards flexible teamwork. *JAIR*, 7:83–124, 1997.
13. T. Vicsek, A. Czirok, E. Ben-Jacob, I. Cohen, and O. Sochet. Novel type of phase transition in a system of self-driven particles. *PHYS REV LETT.*, 75:1226, 1995.