Monopolies Can Exist in Unmanned Airspace

Scott Forer
University of Nevada, Reno
sforer580@nevada.unr.edu

Logan Yliniemi
University of Nevada, Reno
logan@unr.edu

ABSTRACT

With the increased use of unmanned aerial vehicles (UAVs) for both commercial and private use comes the inevitability that oversaturated airspaces will exist. If an airspace becomes congested and difficult to traverse, the possibility of an entity abusing, controlling, and even monopolizing the space can be extremely dangerous. In this paper we show that this type of monopolization can exist. We use cooperative coevolutionary algorithms to examine multiple teams of UAVs coexisting in the same airspace. Considering two equally-sized teams: A and B, if Team A chooses to cooperate with Team B, and considers team B's losses as its own, the system can work fluidly. If Team A chooses to focus on its own concerns while ignoring impacts on Team B, Team B can suffer a 99% increase in midair conflicts. If Team A chooses to actively prevent Team B from fluid operation, Team B's number of midair conflicts can suffer a 394% increase.

CCS CONCEPTS

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

KEYWORDS

Unmanned Air Traffic Management; Coevolution

ACM Reference format:
Scott Forer and Logan Yliniemi. 2017. Monopolies Can Exist in Unmanned Airspace. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 8 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Cooperative multiagent systems (CMASs) have been a growing topic over the last 20 years, yielding many breakthroughs and achievements [5, 7, 17]. There has been sufficient success with a number of multiagent system (MAS) domains; one of which is navigating around obstacles to achieve a unified goal, while systematically learning how to improve their efficiency [13, 14]. For example, the implementation of CMASs are being used to develop policies for unmanned aerial vehicles (UAVs) for search and rescue applications. Allowing the agents to work with each allows faster location of the assets, improving the agent's individual efficiency in addition to the overall system efficiency. This strategy of having the agents and the overall system improve their efficiency simultaneously has largely been explored with CMASs. The issue with this is that there is an assumption that the agents will work with each other to improve the efficiency of the overall system.

The United States National Aeronautics and Space Administration (NASA) is developing an Unmanned Traffic Management (UTM) system to develop a low-altitude airspace for unmanned aircraft system operations [9, 10]. This is in direct response to the rise in popularity of the use of UAVs. Entities within NASA's UTM system would submit a flight plan containing the time of departure, type of aircraft, planned travel speed, and waypoints. The UTM system will check that the proposed flight plan is conflict-free and then tracks the aircraft via GPS. Though the UTM system will check for potential conflicts prior to authorizing a flight plan, it is not capable of distinguishing the difference between a flight plan whose purpose is to simply make it to its destination and an obstructive flight plan. Since the UTM system handles the flight plan request and execution autonomously, there is no fail-safe that understands the purpose of any particular flight plan. This leaves a door open for an entity to submit flight plans that could directly impact the flight plan of another entity. If we consider this on a larger scale, then a team or corporation could submit flight plans that negatively impact other teams or corporations.

An apparent example of this scenario is when we consider multiple drone delivery companies all operating in one airspace. Companies, such as Amazon, are currently working towards deploying swarms of UAVs to deliver packages to their customers; however Amazon needs to wait until regulatory support is in place [1, 2]. NASA's UTM system is the regulatory support that will enable not only drone delivery, but agricultural applications, commercial photography, surveillance, and search and rescue operations. Once the regulatory plans are in place, other companies will enter as entities in the drone delivery market. If an entity looks to exploit this market, they can saturate the airspace with obstructive flight plans and drive their competitors out.

The major contribution of this work is to show that under NASA's current UTM system, an individual or group ("entity") can take actions to create severe congestion in an otherwise healthy airspace, forcing up to 394% more collision avoidance maneuvers than were required otherwise.

The remainder of this work is organized as follows: Section 2 provides the necessary background. Section 3 describes our experimental approach. Section 4 describes the experimental domain. Section 5 provides our experimental results. Finally, Section 6 concludes this work and provides directions of future research.

2 BACKGROUND

MASs are a useful way of developing a system where multiple entities make decisions that effect themselves, other agents, or the

GECCO '17, Berlin, Germany

123-4567-24-567/08/06…$15.00
DOI: 10.475/123_4

environment [5]. In a MAS, agents are rewarded for decisions that they make and how it affects themselves, other agents, or their environment. Agents can be rewarded for making it to their destination, observing points of interest (POI), avoiding conflicts with other agents, or even producing conflicts with other agents.

In a CMAS, agents work together in a team to achieve an individual or global goal [11]. In a sense, agents who perform well with other agents by reducing conflicts, or discovering a high number of POI are rewarded. Inversely, the competitive side of MASs exists where agents receive rewards based how often other agents had conflicts or if the other agent's number of observations of POI are minimized.

Typically models are developed based on whether or not agents in a system are working with or against each other. However, this idea of cooperative or competitive MASs neglects a scenario where agents are not being rewarded in a competitive sense, but rather the agents are focused on optimizing themselves regardless of how it affects other agents or the environment. This middle ground can be modeled by having multiple CMAS teams, where each team has their own objective. This scenario of agents learning cooperatively within their own respective team, however not necessarily with agents form other teams, can better be described as a multi-cooperative multiagent system (MCMAS), which is illustrated in **Figure 1**. A CMAS represents how agents from single team work with one another, where the MCMAS represents how teams of CMASs can exist in the same environment but have the ability to learn differently from one another. For instance, one team may be learning to cooperate with all agents in the system, while another team may only be learning cooperatively with agents in it's own team. In this paper, we use this model to show how a team of non-cooperative agents can harm another cooperating team without being directly rewarded. We also look into how a team of malicious agents can harm a cooperating team when receiving direct rewards.

A common learning algorithm for training agents is an evolutionary algorithm (EA). The multiagent implementation of this comes in the form of a cooperative co-evolutionary algorithm (CCEA). Like a standard EA, agents have multiple solutions or policies, but specific to a CCEA, each policy for each agent is given an overall quantitative value for how well the agents' policies worked together. CCEAs tend to have a population of solutions that converge to a stable set

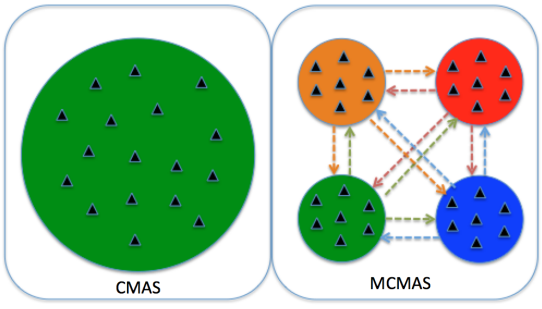

Figure 1: In a CMAS, all agents are cooperating. In a MCMAS, multiple teams of agents are cooperating with their teammates, but not necessarily with other teams.

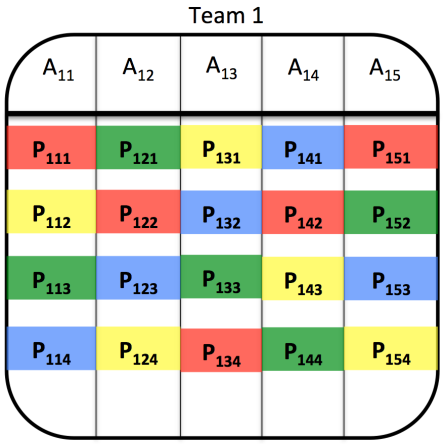

Figure 2: *Random Team Builder* - A CCEA randomly selects a policy for each agent to build a simulated team.

of solutions and not necessarily the optimal solution. Leniency is a form of a credit assignment that helps a CCEA train the population not only faster, but it allows a more optimal population of solutions to be obtained.

2.1 Cooperative Co-evolutionary Algorithms

A CCEA is a method used to simultaneously train multiple agents to work together cooperatively to achieve a global goal. This consists of a team of multiple agents, where each agent has multiple policies. The policies contain a set of decisions that directly effect that policy's fitness. A key difference between an EA and a CCEA is that a CCEA allows multiple agents to learn to produce a stable solution space rather than the most optimal. This is extremely useful for MAS exploration and path planning problems [8].

Since these policies need to work well with one another, they need to be graded as a team. This random selection repeats until all of the policies for each agent are selected and assigned a fitness. **Figure 2** shows a visualization for a single team of agents and how policies are selected to build a "simulated team" where each of the different colors represent a different simulated team.

As a way to determine which policies for each agent were the best, each agent performs its own EA. Similar to a standard competitive style EA, two policies are randomly selected and compared. The losing policy becomes a copy of the winning policy and is given a slight mutation. This process is repeated until the population of policies for each agent contains one half corresponding to the winners and the other half corresponding to the mutated copies of the winners. Once each agent has performed an EA on their policies, the next generation of the CCEA begins and randomly selected policies are simulated again.

2.2 Leniency

CCEAs are great at producing coorperative polices, however they have a difficult time when also trying to optimize the global fitness. Each policy of each agent is only simulated once during a generation and it's fitness is directly affected by the other agents' policies that it is simulated with. Leniency addresses the problem that one policy

could have performed better if it is paired with a different set of policies. It is a method that allows each policy to be evaluated on several different simulated teams before assigning it a final fitness for that generation [12]. The most simplistic implementation of leniency is to allow each policy to be simulated in different teams, where it's fitness would be the fitness associated with the best team that it was on. This allows the policies to not only learn to work with each other, but drive the solution space towards the optimal solution.

2.3 Related Work

In the case of multiagent systems, it is largely assumed that the agents will work cooperatively with one another since it is far easier to control a system where there is a unified goal. However, CMASs are not always the appropriate model to use. The Tragedy of the Commons is an economic theory where multiple individuals working independent of each other deplete a finite resource. An example of this is how fish populations in the oceans have been decimated because people have interpreted the "freedom of the seas" to include a right to fish them without any quantity restrictions [6]. Much like people overfishing, an entity could interpret the drone delivery market as the freedom to schedule as many flights as they want, without considering others uses for the airspace.

This issue where airspace becomes a finite resource is amplified when the agents can explore it freely without consequences. Colby, Knudson, and Tumer explored how a MAS could be trained such that the agents would avoid each other while being able to navigate through an airspace to find POI [3]. By using a CCEA to train a neural network for each agent, 92% POI coverage can be achieved while providing zero conflicts between agents.

With the inclusion of a finite resource into a MAS brings with it the possibility of competition. Just like in nature where predators and prey evolve over time, a competitive MAS can also evolve [15]. By simultaneously co-evloving two teams to compete against one another, an arms race develops where each team is being negatively affected by the other teams in the system. This competitive scenario is applicable to airspace as it becomes more and more saturated.

Hall of Fame (HOF) was introduced by Rosin and Belew for competitive co-evolution originally, where the best individuals are saved to be tested against later generations [16]. This allows the policies of the individuals selected for the HOF to contribute to future generations, and that new individuals may be tested against the HOF policies. This method has been extended to CCEAs where the concept of a HOF team can be implemented [4]. In this work, leniency was used in place of HOF as it is able to achieve a desirable learning speed. HOF will be implemented in future work.

3 TEAM-BASED CO-EVOLUTIONARY ALGORITHM

In this paper we investigate how two teams interact given various parameters which make them fully cooperative, non-cooperative, or even malicious. The team-based CCEA's are set up in such a way that a single policy from each agent in each team is randomly selected and simulated. More simplistically, we can imagine that **Figure 2** is extended to include the agents and policies of the other team as shown in **Figure 3**.

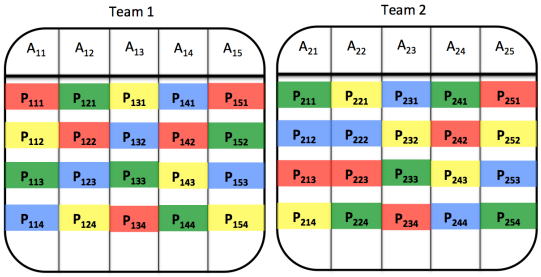

Figure 3: *Random Multi Team Builder* - Multiple CCEAs randomly select a policy for each agent for each team to build multiple simulated teams.

Algorithm 1 describes how the CCEA is implemented in this paper. Line 1 creates a random set of polices for each agent in both teams. The selection process then works the same as a standard CCEA, where a policy from each agent from both teams is randomly selected in line 5, which creates the simulated team. Line 6 is described in more detail in **Algorithm 2**. To obtain the fitness for each policy, all of the policies on Team 0 would have the same fitness value as each other and all of the policies on Team 1 would have the same fitness values as each other, which is performed in line 7 in **Algorithm 1**. This process is repeated, until each policy has been assigned a fitness value only once. When leniency is implemented, this process is repeated n number of times and a policy's fitness is reassigned if it is able to perform better on a different simulated team.

The next portions of the CCEA used in this paper are the Down Select, Repopulate, and Mutate functions, lines 11, 12, and 13, which are described in the Section 2.1 of this paper. Following that the conflict and fitness data corresponding to the best polices for each agent in both teams are stored, the entire process from line 5 to 14 is repeated for m number of generations.

Both leniency and HOF allow for the CCEA to better learn to produce optimal solutions, however HOF was out of the scope of this project and will be implemented in future work. In this work, leniency is selected because it allows for a better representation of how well each individual policy performs on different simulated teams. Leniency also helps to speed up the learning process substantially. Each simulated policy is assigned a fitness value that is dependent on how well it performed, given a preset scenario.

4 UAV AIR CONFLICT DOMAIN (UCD)

The basic parameters for the UAV Air Conflict Domain (UCD) are set in such that an x by y by z unit space will be highly saturated with UAVs. By introducing n number of agents into the environment, the airspace becomes very difficult to navigate without triggering the crash avoidance systems while still allowing the agents to learn to not cause a conflict with one another.

In each simulation, agents are given a set of w number of waypoints which include their starting and ending waypoints. During the simulation, each agent has set amount of time to move from waypoint to waypoint. Once an agent has made it to their final waypoint, their conflict data and flight time is stored. The conflict data for an agent is then used to judge how well that set of waypoints worked

Algorithm 1: CCEA Algorithm. The CCEA is populated by vector teams ($\mathcal{T}$) which contain a vector of agents ($\mathcal{A}$), which contain a vector of polices ($\mathcal{P}$) and randomly builds teams for the the simulator ($\mathcal{ST}$), which returns a fitness for each policy ($\mathcal{F}_p$).

Input: $T, A, P, Num\_T, Num\_A, Num\_P, Gen\_max, Len$
Output: $\mathbb{F}_t$

```
Initialize_Population(T, A, P)
for Gen < Gen_max do
    for Amount lenient L en do
        for Each Policy P do
            ST ←Build_Team_For_Simulation(P)
            Simulate_Team(ST)
            Get_Team_Fitness(ST)
    for Each Team T do
        for Each Agent A do
            for Num_P/2 do
                Down_Select(F_p)
                Repopulate(P)
                Mutate(P)
    Store_Data(F_t)
Return Fitness_Data F_t
```

at avoiding conflicts and allowing the agent to make it to it's final destination.

Algorithm 2 describes the simulation process used in the UCD. Each agent begins the simulation at their first waypoint, which is performed in line 3. For each agent, line 6 calculates where it will be at the end of each time step. Once each agent has their projected state, then line 8 is performed, which is detailed in **Algorithm 3**. If there exists a potential conflict between two or more agents, then all agents involved in the conflict will have their speeds reduced, changing their projected state. Line 10 of **Algorithm 2** simply looks to see if an agent has reached it's target waypoint. If the projected state predicts that an agent will overshoot the target waypoint, the agent will adjust it's speed as needed to not overshoot. Once the agent makes it to their target waypoint, it is assigned a new target waypoint. Lastly line 11 assigns an agent's current state to be their projected state. This process is repeated until the allotted time given for a simulation runs out.

Given any agent's projected state, the simulator calculates the separation distance between each agent, checks for potential conflicts, and performs a crash avoidance if necessary. The crash avoidance used in this domain is detailed in **Algorithm 3**. Since a potential conflict could occur between two or more agents' current states and their projected states, line 2 gets the incremented states for an agent between it's current state and projected state at k time interval. If the separation distance calculated in line 5 is less than the acceptable crash avoidance distance, the crash avoidance functions will take place represented by lines 7 through 12 where the fitness is determined by the behavior of the teams involved in the potential conflict. Line 10 then adds a value of one to the number of conflicts for that agent's team. In this domain, the conflict data is representative of

Algorithm 2: `Simulator Algorithm`. The Simulator ($\mathcal{S}$) is populated by vector of agents in a simulated team, ($\mathcal{ST}$) where each agent has a vector of waypoints ($\mathcal{W}$) and returns a fitness for each agent ($\mathbb{F}$).

Input: ST, W, S_{time}
Output: $\mathbb{F}_i$

```
1  Set_Initial_Conditions(S)
2  for iter = 0 → size(ST) do
3      ST ← Initialize_Agents(W)
4  for Each time step do
5      for Each agent in simulated team ST do
6          Get_Projected_States(W)
7      for Each agent in simulated team ST do
8          Check_For_Conflicts(W)
9          Track_Conflicts(W)
10         Check_If_At_Target_Waypoint(W)
11         Get_New_States(W)
12 Return Agents Fitness F_i
```

Algorithm 3: `Crash Avoidance Algorithm`. Each agent's projected state is compared to other agents to check for possible conflicts in the Simulator ($\mathcal{S}$). If a potential conflict exists the conflict matrix is evaluated and penalties are given based on the experiment being conducted ($\mathcal{E}$).

Input: ST,
Output: $\mathbb{F}_m$

```
1  for Each agent in simulated team ST do
2      Get_Incremented_Projected_State(ST)
3  for Each agent in simulated team ST do
4      for k < Time_Step do
5          Get_Separation_Distance(ST)
6          if Separation_Distance < Crash_Avoidance_Distance
             then
7              Set_Fitness_Matrix(E)
8              Calculate_Fitness_Matrix(ST)
9              Penalize_Agent(ST)
10             Store_Conflict_Data(ST)
11             Change_Agent_Travel_Speed(ST)
12             Get_New_Projected_State(ST)
13 Return Fitness F_m
```

the global fitness for each team. Lastly the agents involved in the potential conflict will have their travel speeds for that time step reduced and a new projected stated will be calculated. This reduced speed is to represent the how an agent would have to move off a straight line path to avoid a crash.

4.1 Fitness Calculations

In the UCD, several fitness calculations are considered to model how two teams can learn to optimize their population of solutions differently. **Figure 4** illustrates how fitness is calculated based on

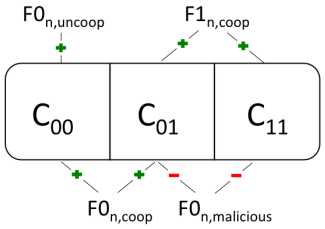

Figure 4: - Conflict Matrix: In the cooperative, uncooperative, and malicious cases, Team 0's fitness calculation changes, but Team 1's is always done cooperatively (Equations 2–5).

the type of learning that caused the conflict. The fitness matrix allows agents to be penalized or rewarded depending on if their team is cooperatively, uncooperatively, or maliciously learning. C_{00} is a conflict between two agent on Team 0, C_{01} is a conflict between an agent on Team 0 and an agent on Team 1, and C_{11} is a conflict between two agent on Team 1.

The conflict matrix allow agents in different teams to be penalized or rewarded based on actions that caused conflicts between agents. The UCD keeps track of conflicts that involve only agents in Team 0 denoted by C_{00}, agents in Team 0 that conflict with agents in Team 1 or vice versa, denoted by C_{01}, and conflicts that only involve agents in Team 1 denoted by C_{11}. The fitness for each team is the sum of the fitnesses for each agent in that team. Agent fitness is the sum of penalties and rewards for each agent in that team for each time step throughout the simulation. The fitness calculation for a given team is:

$$\mathcal{F}_t = \sum_{n=(0):number\_of\_agents} (\mathcal{F}_n) \tag{1}$$

Where F_t is the fitness of each team which is determined by the fitness of each agent in that team, note that agents on Team 0 are capable of learning with various behaviors where agents on Team 1 always learn cooperatively. F_n is the fitness for the n^{th} individual.

Two Cooperative Teams - In the case of two cooperative teams, agents on any given team are penalized if one of it's agents cause a conflict regardless of which team the other agent is on. From the conflict matrix, agents on Team 0 receive penalties when an agent from Team 0 conflict with another agent from Team 0 or conflict with an agent on Team 1. Agents on Team 1 are penalized when a conflict occurs between two agent on Team 1 or between an agent on Team 1 and Team 0. The fitness calculation for Team 0 and Team 1 where $F0_{n\_coop}$ is the fitness for the n^{th} individual on Team 0, $F1_{n\_coop}$ is the fitness for the n^{th} individual on Team 1 is:

$$\mathcal{F}0_{n,coop} = \sum_{t=(0):max\_time} (C_{00} + C_{01}) \tag{2}$$

$$\mathcal{F}1_{n,coop} = \sum_{t=(0):max\_time} (C_{01} + C_{11}) \tag{3}$$

One Uncooperative and One Cooperative Team - In the case of one uncooperative team and one cooperative team, the uncooperative team is only penalized if one of its agents conflicts with another one of its agents. In this domain, Team 0 is the uncooperative team and therefore is only penalized through the conflict matrix when agents on Team 0 cause a conflict with another agent on Team 0. The cooperative team is Team 1 and is penalized each time any of its agents are involved in conflict regardless of which team the other agent in on. The fitness calculation for Team 0 is shown below, where $F0_{n\_uncoop}$ is the fitness for the n^{th} individual and the fitness for agents on Team 1 is defined by $F1_{n\_coop}$.

$$\mathcal{F}0_{n,uncoop} = \sum_{t=(0):max\_time} (C_{00}) \tag{4}$$

One Malicious and One Cooperative Team - The last case that is considered in this domain is when one team acts maliciously towards another team. The malicious team in this domain is Team 0. The malicious team receives a reward every time an agent from Team 0 conflicted with an agent from Team 1 or when two agents from Team 1 conflicted with each other. Agents on Team 0 are never penalized for being involved in a conflict. Similar to the other two cases, Team 1 is still penalized for any conflicts that it's agents were involved in. The fitness calculation for Team 0 is shown below where $F0_{n\_malicious}$ is the fitness for the n^{th} individual and the fitness for agents on Team 1 is defined by $F1_{n\_coop}$.

$$\mathcal{F}0_{n,malicious} = \sum_{t=(0):max\_time} (-C_{01} - C_{11}) \tag{5}$$

5 EXPERIMENTAL PARAMETERS

The parameters for this experiment are developed such that the airspace would be difficult to learn in. In each experiment, the basic parameters are keep the same to allow the different experiments to be compared fairly. The basic parameters for each experiment are a 35 by 35 by 35 unit airspace, 28 agents divided evenly into two teams, and 150 seconds of allotted time to make it to all of their waypoints. Each agent has 50 policies, where each policy contains 8 waypoints and are bounded to always be within the airspace. Each agent has a normal travel speed of 5 units per second and a crash avoidance speed of 2.5 units per second. The crash avoidance radius is set to be 5 units and agent movements through the airspace are calculated every 0.1 seconds. For the experiments that use leniency, each policy is allowed 5 chances to find a random team to perform well on. The learning algorithm is to be ran for 300 generations where each waypoint for a copied policy has a 50% chance of being mutated in the x, y, and z direction between [-1, 1] units. Lastly, each experimental result is the average of 30 statistical trials and the error was calculated by the standard deviation for each generation dived by square root of the number of statistical trials ($\sigma/\sqrt{N}$).

To show that one team can harm another, the following experiments were conducted (Sections 5.1–5.4, respectively):

- **Two Teams Same Size With Cooperative Learning:** Serves as a baseline and shows that given a highly saturated airspace, teams can learn to avoid conflicts.
- **Two Teams Same Size With Uncooperative Learning:** One team is cooperative and one team is uncooperative. Team 0 learns to avoid internal conflicts with other agents from Team 0, but not conflicts with Team 1.
- **Two Teams Same Size With Malicious Learning:** Team 0 will learn cause conflicts with agents in Team 1.
- **Static Waypoints:** Team 1 will learn policies that reduce the number of internal

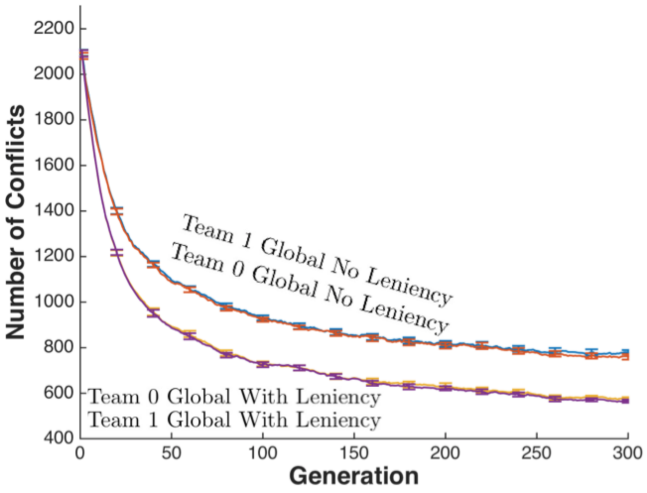

Figure 5: Conflict data for two teams learning cooperatively. Because the teams were both cooperative and equally sized they experienced a similar number of conflicts in each experiment. Both teams were able to learn substantially better with leniency.

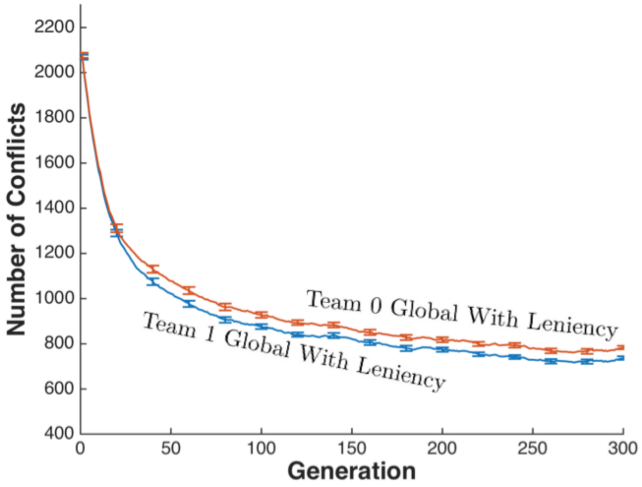

Figure 6: Conflict data for two teams with one cooperative and one uncooperative team. Both teams experienced more conflicts than that of the cooperative case shown previously.

conflicts, which are then kept static as Team 0 is introduced into the environment, learning in different cases: cooperatively, uncooperatively, and maliciously.

5.1 Two Teams Same Size Cooperative Learning

The purpose of the first experiment conducted is to show that two teams that work cooperatively with one another can learn to minimize the amount of potential conflicts. The goal is to have each team statistically learn at the same rate and achieve the same number of conflicts. Additionally, we want to show that when leniency is included, the two teams are able to learn faster and achieve a better final number of conflicts. Lastly this experiment serves as a benchmark for the other experiments as it presents an ideal learning curve for this domain when two teams are learning to navigate a highly saturated airspace.

From **Figure 5** we can see that both teams are able to minimize their number of potential conflicts. Both teams are also able to learn at the same rate and have average final number of conflicts of 776.16 and 760.00 that are within error bars of each other and are therefore statistically the same graph. This result also acts as a verification for our domain, such that instead of one large team working cooperatively with one another, we can now have two teams working cooperatively with each other. Both teams learn at the same rate and achieved the same final number of conflicts; this is significant since each team has different global evaluations. Lastly we can see that when the teams are able to learn with leniency, they are able to learn faster and achieve better average final conflict values of 574.90 and 565.36.

5.2 Two Teams Same Size Uncooperative Learning

In the second experiment, we explore how both teams learn to avoid potential conflicts when one team was acting uncooperatively with the other. In this case agents on Team 0 are only penalized for being involved in a conflict with another agent on Team 0. However, agents on Team 1 are still penalized for being involved in conflicts with agents from Team 0 or with another agents on Team 1. The goal is to show that when agents on Team 0 are acting uncooperative with agents on Team 1 and a conflict between the two teams occurs, then agents on Team 0 will keep their flight path and force agents on Team 1 to take a different one.

Figure 6 shows that when agents on Team 0 are learning uncooperatively with agents on Team 1, then Team 1's global conflicts are only 6.31% higher than that of Team 0's global conflicts. Due to Team 0's uncooperative learning, Team 0's and Team 1's average global number of conflicts increase from 574.90 and 565.36 to 736.95 and 783.47 in comparison to when both teams were learning cooperatively. Since Team 0 learns in an uncooperative manner, Team 1's final global number of conflicts increases by 38.57%. This rise in the number of conflicts is directly due to the conflict counter keeping track of all of the conflicts between agents regardless of how the team learns.

5.3 Two Teams Same Size Malicious Learning

Another method that a team of agents can use to learn to interact with agents from another team is to maliciously attack them. In this experiment, agents from Team 0 are given a reward for causing conflicts that involved agents from Team 0 and Team 1. In addition to this, agents from Team 0 are also rewarded for conflicts between two agents on Team 1. Agents on Team 1 are trained to learn cooperatively and therefore are penalized for being involved in any conflicts. By running this scenario, we are able to see if a team of agents could learn to optimize their flight path in a hostile environment.

Figure 7 provides evidence that a team of agents acting maliciously can greatly impede the learning progress of another team learning to avoid potential conflicts. In this experiment agents on Team 0 achieve an average final number of conflicts of 1635.20,

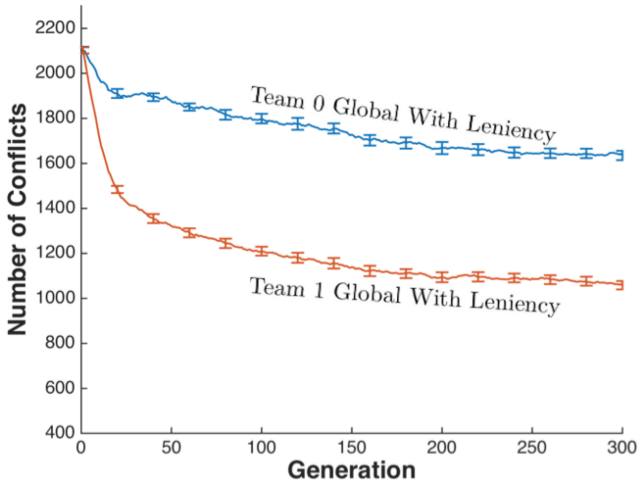

Figure 7: Conflict data for one cooperatively learning team and one maliciously learning team. Both teams experienced substantial increases in conflicts when compared to the cooperative case or uncooperative case shown previously.

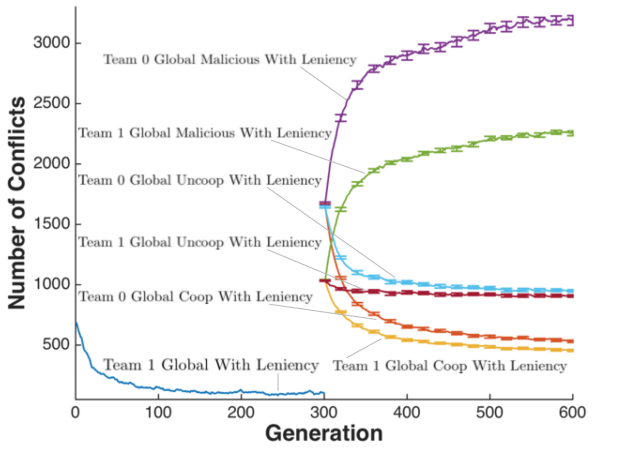

Figure 8: In the first 300 generations, Team 1 learns to minimize their conflicts. After 300 generations Team 1's behavior is held constant and Team 0 is introduced and learns with leniency from generations 300-600. Team 0 learning cooperatively produces the fewest conflicts; uncooperative produces moderately more conflicts; malicious produces far more conflicts.

where agents on Team 1 achieve a average final number of conflicts of 1058.60. If we compare these results to that of the strictly cooperative case, we see that the number of conflicts greatly increases when agents on Team 0 learn to maliciously attack agents on Team 1. Due to Team 0's malicious behavior, Team 1's average final number of conflicts increases by 87.24%.

5.4 Static Waypoints

Although the other experiments shed light on what happens when both teams are learning simultaneously, they do not provide any insight as to what would happen if only one team is able to learn. In this experiment we first optimize the flight path for agents on Team 1, then introduce Team 0 into the environment. We then run the cooperative, uncooperative, and malicious cases for Team 0 and keep agents on Team 1 from learning to change their optimized flight path. The purpose of keeping agents on Team 1 form learning to alter their flight path is to show a more realistic way that a team can harm another team by simply having their agents learn to produce flight paths based on a learning algorithm.

Figure 8 shows the average conflict data for both teams for the cooperative, uncooperative, and malicious cases as well as the learning curve for when the environment only consist of Team 1. Before introducing both teams to the environment, agents on Team 1 are allowed to optimize their flight path to an average final number of conflicts of 101.09.

In one scenario, agents on Team 0 learn to cooperatively work with agents on Team 1. This cooperative behavior allows for Team 0's average final number of conflicts to be reduced to 531.48. Although the agents on Team 1 are not allowed to deviate from their optimized flight path, Team 0 is learning to cooperate and this causes the average final number of conflicts for Team 1 to be reduced to 455.52. The reason that there is such a large increase from when there was just Team 1 in the environment to when a cooperative Team 0 was introduced is due to the introduction of 14 more agents into the environment, causing the airspace to be more saturated and difficult to learn in.

For the scenario where an uncooperative Team 0 is introduced and agents on Team 1 are only allowed to used their optimized flight path, we see a statistical increase in the average number of conflicts. Due to Team 0's uncooperative learning, Team 1 is only able to achieve an average final number of conflicts of 906.94, which is a 99.10% increase to that of the cooperative scenario.

The last scenario in this case is a malicious Team 0 being introduced into the environment after agents on Team 1 optimize their flight path. This simple behavior change causes a very drastic increase in the average number of conflicts for agents on Team 1, with an average final number of conflicts of 2253.7, which is a 394.75% increase from the cooperative scenario.

5.5 Discussion

The experiments presented in this paper have allowed us to look at airspace congestion from multiple aspects. The first experiment where two teams of the same size learned to cooperate with one another proved that even though an airspace may be highly saturated, as long as all the agents in the environment learn to work together, the number of potential conflicts can be greatly reduced. Since both teams' learning curves were statistically the same, we can say that the cooperative learning process worked. Lastly, the first experiment showed how the CCEA was able to learn not only faster but better by using leniency.

The next experiment explored how a team may want to learn differently than the other. When Team 0 learned in an uncooperative manner and Team 1 was still trying to learn cooperatively, there was a statistically drastic increase in the number of potential conflicts

Table 1: Average Final Number of Conflicts Two Teams Learning

Learning Behavior	Team Learning	Team 0	Team 1
Cooperative no Leniency	0 & 1	776.16	760.00
Cooperative with Leniency	0 & 1	574.90	565.36
Uncooperative	0 & 1	736.95	783.47
Malicious	0 & 1	1635.20	1058.60

Table 2: Average Final Number of Conflicts One Team Learning

Learning Behavior	Team Learning	Team 0	Team 1
Optimized Team 1	1	N/A	101.09
Cooperative Team 0	0	531.48	455.52
Uncooperative	0	948.90	906.94
Malicious	0	3188.4	2253.7

by an average of 38.57%. Since Team 0 was only learning to avoid potential conflicts with its own team members, the airspace became more difficult for Team 1 to learn in. Whenever there was a potential conflict between the two teams, Team 1 was forced to change their flight plan, where the Team 0 was able to keep theirs.

The third experiment investigated how the two teams learn when Team 0 was maliciously trying to cause conflicts with Team 1 while Team 1 was still trying to learn cooperatively. This behavior made it even more difficult for Team 1 to learn to reduce the number of conflicts and we saw an increase in the average by 87.24%. **Table 1** lists the average final number of conflicts for each behavior when both teams are capable of learning.

In the final experiment, we identified the problem caused when only Team 0 is able to learn to avoid or cause conflicts in an already optimized airspace. After Team 1 was allowed to optimize its flight path, Team 0 was introduced. The newly introduced team was then able to learn cooperatively, uncooperatively, or maliciously. In the cooperative scenario, Team 0 was able to learn to optimize their flight path as much as possible. The uncooperative scenario showed that there was an increase in potential conflicts by an average of 99.10%. For the malicious scenario, there was an increase in the number of conflicts by 394.75%. **Table 2** lists the average final number of conflicts for each behavior when only one team is capable of learning.

6 CONCLUSION

In this work we investigated the use of multiple multiagent systems to model competitive entities using the UTM system. Because of the broad increase in drone usage in recent years and the desire for drone-based package delivery to become a mainstream service, it is an inevitability that multiple entities will be working within a shared, limited airspace.

The experiments, with two equally-sized multiagent teams with one team always working cooperatively and the other team working either cooperatively, uncooperatively, or maliciously, show how the different entities using the UTM system will interact and how it will change the efficiency throughout of the system. If an entity chooses to be malicious toward the other team's smooth operation, this can lead to a 394% increase in the number of potential mid-air conflicts in comparison to a strictly cooperative airspace.

Even in the best-case scenario, where each team is aware that it should continue to optimize its own paths, a team choosing to act in their own best interests instead of the overall system's best interests can lead to a 38% increase in potential mid-air conflicts. This is a substantial loss in performance for other stakeholding entities.

Future work on this topic includes implementing no-fly zones (modeling areas such as airports or congested pedestrian urban centers), which will create a more difficult routing problem and possible bottlenecks. Additionally we will examine how each team learns when team sizes are varied. If an entity can create a large negative impact in a saturated airspace with even a very small number of agents, this may drive the necessity of additional safeguards.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NNX15AI02H issued through the Nevada NASA Space Grant Consortium.

REFERENCES

[1] Amazon. 2015. Amazon Prime Air. (July 2015). http://www.amazon.com/b?node=8037720011
[2] Amazon Corporation. 2015. *Revising the Airspace Model for the Safe Integration of Small Unmanned Aircraft Systems*. Technical Report. Amazon White Paper.
[3] Mitchell Colby, Matt Knudson, and Kagan Tumer. 2014. Multiagent Flight Control in Dynamic Environments with Cooperative Coevolutionary Algorithms. *AAAI* (March 2014), 110–115.
[4] Mitchell Colby and Kagan Tumer. 2012. Shaping Fitness Functions for Coevolving Cooperative Multiagent Systems. *AAMAS* 1 (2012), 425–432.
[5] Claudia V. Goldman and Shlomo Zilberstein. 2003. Optimizing Information Exchange in Cooperative Multi-agent Systems. *AAMAS* (2003), 137–144.
[6] Garrett Hardin. 2009. The Tragedy of the Commons. *Journal of Natural Resources Policy Research* 1, 3 (2009), 243–253.
[7] Nicholas R. Jennings. 2000. *On agent-based software engineering*. Artificial Intelligence, Vol. 117. Elsevier B.V.
[8] Matt Knudson and Kagan Tumer. 2010. Coevolution of Heterogeneous Multi-Robot Teams. *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (2010), 127–134.
[9] NASA. 2015. NASA UTM. http://utm.arc.nasa.gov/index.shtml. (December 2015). http://utm.arc.nasa.gov/index.shtml
[10] NASA. 2015. *UTM: Air Traffic Management for Low-Altitude Drones*. Technical Report NF-2015-10-596-HQ. National Aeronautics and Space Administration.
[11] Liviu Panait and Sean Luke. 2005. Cooperative Multi-Agent Learning. *AAMAS* 11 (November 2005), 387–434.
[12] Liviu Panait, Keith Sullivan, and Sean Luke. 2006. Lenient Learners in Cooperative Multiagent Systems. *AAMAS* (May 2006), 801–803.
[13] M. V. Nagendra Prasad, Susan E. Lander, and Victor R. Lesser. 1996. Cooperative Learning over Composite Search Spaces: Experiences with a Multi-agent Design System. *AAAI* (1996).
[14] M. V. Nagendra Prasad, Victor R. Lesser, and Susan E. Lander. 1996. Learning Organizational Roles in a Heterogenous Multi-agent System. *AAAI* (1996).
[15] Padmini Rajagopalan, Aditya Rawal, and Risto Miikkulainen. 2010. Emergence of competitive and cooperative behavior using coevolution. *GECCO* (July 2010), 1073 – 1074.
[16] C. Rosin and R. Belewe. 1996. New methods for competitive coevolution. *Evolutionary Computation* (1996), 1–29.
[17] Milind Tambe and Paul S. Rosenbloom. 1996. *Architectures for Agents that Track Other Agents in Multi-Agent Worlds*. Intelligent Agents, Vol. 1037. Springer Berlin Heidelberg.