Emergent Behaviors of Multi-Objective Multi-State Swarms in Dynamic Underwater Scenarios

Jon H. Roach, Benjamin B. Thompson, and Robert J. Marks II

***Abstract*—The allocation of resources between tasks within a swarm of agents can be difficult without a centralized controller. Disjunctive control has been shown to be a viable method to control the behavior of a swarm. In this project, a disjunctive fuzzy control system is used to solve the problem of resource management. Multi-state swarms are evolved with an offline learning algorithm to adapt to dynamic scenarios with multiple objectives. Some of the emergent behaviors developed through the evolutionary algorithm were state-switching and recruitment techniques.**

***Index Terms*—Keywords: swarm intelligence, multi-state, task switching, fuzzy control, emergent behavior**

I. Introduction

An important component of swarm intelligence systems is division of labor. If there are multiple, possibly competing, objectives, how is a group of autonomous units able to decide how many units should work on each objective? In this paper, we will demonstrate the ability of these swarms to adapt to dynamic conditions by autonomously reallocating resources as necessary in order to achieve multiple objectives. Our solution is based on strategies found in nature, both the state-switching methods employed in ant colonies and recruitment techniques found in swarms of bees [1]. These methods are tested in simulations that require the swarms to accomplish two objectives at the same time: such as defending a friendly unit or attacking enemy targets. Section II provides a brief background of Swarm Intelligence. In Section III, we describe a Point Attack & Point Defense game played by two swarms, in which each works to both defend its base and destroy its enemy's base. Agents use threshold functions to control state-switching behavior. Section IV describes a Search & Destroy mission that a swarm is tasked with completing. The swarm must find and destroy an enemy target within a time limit using recruiting techniques. These two scenarios are combined in a Base Attack simulation, which is described in Section V. Here, both state-switching thresholds and recruiting methods are used by the swarm. In each case, an evolutionary learning algorithm is used to optimize these strategies based on fitness scores. The resulting emergent behaviors are shown to be robust as the swarms continue to perform well even as the population of the swarm decreases. A portion of the following results have previously been published [12].

II. Swarm Intelligence

In a swarm, each agent is computationally simple, compared to the complexity of the whole. Individual agents follow a set of simple rules which define the agent's behavior. However, when a large number of the agents are allowed to work together, the result can be a unique and sometimes surprising emergent behavior. For the following simulations, decisions the agents make, such as "Where do I go next?", or "Should I begin working on a new task?" are controlled via inputs from a group of sensors. These inputs are fed into weighting functions which determine the resulting decisions of the unit.

Previous research [3][6] has focused on designing swarms with a single objective. These swarms demonstrated the use of Combs control [6] as a viable solution to determining the individual rules within a swarm. Most previous simulations involved two swarms competing in a simple game. By using an evolutionary algorithm to optimize the fitness scores of these swarms, each swarm was able to develop strategies and counter-strategies to beat its opponent. Our goal throughout this project is to expand upon the previous work to more complex swarms that can achieve two or more objectives in a dynamic environment.

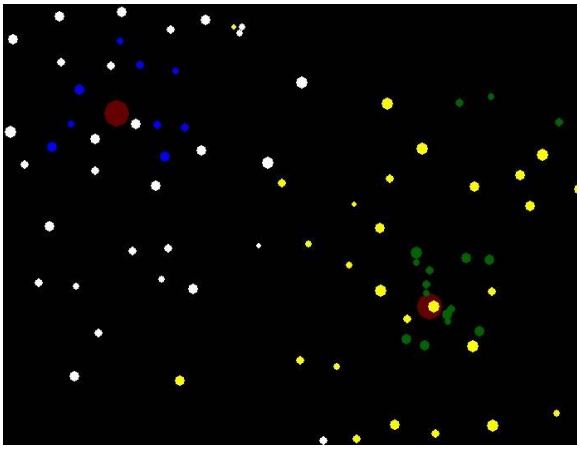

Figure 1. A screenshot from the Point Attack & Point Defense simulation. Swarm one is shown by the blue and white dots, with blue representing defending agents and white, attacking. Swarm two is shown in green and gold for defensive and attacking agents, respectively. Bases are shown in red. The size of the dot is an indicator of the relative strength of the agent.

III. Point Attack & Point Defense Swarms

A. Simulation

The first simulation that we investigate is a Point Attack & Point Defense competition. Two swarms compete against each other. Each swarm has two, possibly competing objectives: to guard its own base from enemy attacks, and to find and destroy an enemy base. The game is played in a rectangular, two-dimensional grid, as shown in Figure 1. The edges of the grid a rigid, so when an agent runs into the edge, it bounces off in the opposite direction.

At the beginning of the simulation, two teams are initialized, which we will refer to as swarm one and swarm

two. Swarm one is initialized on the left wall with swarm two on the right, like many competitive team-based games. When two agents bump into each other, a battle is triggered. In a battle, each of the two agents involved does a random amount of damage to its opponent, scaled by the strength of the agent. When an agent's strength reaches zero after a battle with an opposing agent, it is removed from the playing area. When an agent reaches the enemy base, the same attack algorithm is used. In the case of attacking an enemy base, however, the battle is one sided as the base is not able to fight back. Each team's base is placed in a randomized location near its friendly wall. The base starts with a large strength value. The simulation continues until one of the two bases has been destroyed by reducing its strength to zero. Since each swarm has two objectives (aggressive and defensive), agents are allowed to take on one of two states. Aggressive agents are tasked with finding and attacking the enemy base, while defensive agents repel enemy attacks on the friendly base.

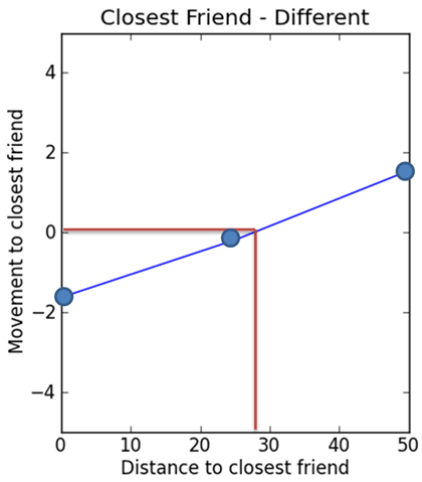

Figure 2. Sample weighting function. This function represents the sensor that finds the distance between an agent and its closest friend of a different state (within range). In this example, an agent within 28 units of the friendly agent is repelled from the agent while an agent at a distance of greater than 28 is attracted to the other agent. The emergent behavior of this simple rule is the agents attempting to maintain a distance of 28 units away from all agents of a different state.

Each agent is able to sense nearby units and can distinguish friendly agents from enemy agents. In addition, agents are aware of the current state of other units they can sense. The sensor readings are fed into weighting functions, which determine the resulting movement of the agent. A sample weighting function is shown in Figure 2. The weighting functions come in pairs. One function defines movement toward or away from the object being sensed, while the second controls movement parallel to the object. The combination of the two allows the agent a full range of movement relative to the object in the two dimensional playing grid. In addition to the contributions of the series of weighting functions, each agent also takes a random step (or twiddle, as we call it). This twiddle is a crucial component to the simulation, since without it, agents out of range of anything else would remain motionless.

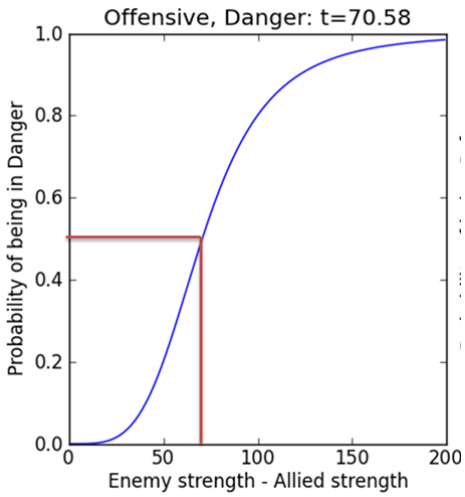

Figure 3. Sample threshold function. This function shows how the difference between enemy strength and allied strength is used to determine whether or not the agent is in danger. In this case, the threshold 70, so if an agent senses that the difference is less than 70, the agent will most likely not consider itself to be in danger. If the difference is sensed to be greater than 70, then odds are the agent does think it's in danger.

The swarms in the Point Attack & Point Defense scenario are multi-state. Agents within the swarms are able to take on one of multiple states in order to allow the swarm as a whole to achieve multiple objectives simultaneously. In addition to being multi-state, these swarms also need to be dynamic for them to be able to adapt to changing conditions within the playing field. The first problem is choosing a method that would allow individual agents to switch tasks, if needed, without the use of a centralized controller. Our solution was inspired by the behaviors of certain types of ants in nature. Within most ant colonies, there are multiple roles the ants fulfill. When circumstances dictate, ants are able to temporarily switch tasks to help the rest of the anthill with a task that needs extra work. For instance, a soldier ant that senses a large amount of food piled up that needs to be taken into the hill could decide to switch and function as a worker until the transportation of the food is completed. The decision making process can be modeled as a threshold function with a sigmoid shape [1].

In a similar fashion, agents within the Point Attack & Point Defense swarms switch tasks based on threshold functions attached to sensors that count the strength of nearby agents in different states. A sample threshold function is shown in Figure 3. If an agent senses a large number of agents around it working on the same task relative to the agent's threshold, role saturation occurs and it is inclined to switch to the other task. Also, if an agent senses the relative strength of enemy units with respect to friendly agents to be too large, it may decide it's in danger and send out a distress signal asking nearby units of a different state to switch to the agent's current state. If those other agents sense that their current task has enough agents and are in a "safe" position, they may be inclined to switch to help out the agent in distress. In the early stages of the design process, agents considering switching counted the number of nearby agents, not their strengths. This caused some problems, as weaker agents were counted the same as stronger agents. For instance, a group of seven weak

agents encountering two strong opposing agents would think that they were in good shape, when in fact, the group of two had a distinct advantage. By comparing the relative strengths of nearby agents, the swarm was better able to judge when agents were safe or in danger.

B. Coevolution

The weighting functions were optimized using an off-line learning algorithm based on coevolution. In coevolution, two populations are evolved against each other. One of the drawbacks of coevolution is that it can become difficult to tell if strategies are improving simply by looking at the results of the fitness function. At first glance it appears that there is not much progress being made, but the populations are learning as the evolution process goes on. Eventually, however, the populations will reach a point of equilibrium, where the teams have both maximized their fitness performance and are equally matched.

Figure 4. The Coevolution Process.

The coevolution process is shown in Figure 4. First, both populations (pop1 and pop2) are filled with randomly generated teams. Then, pop2 is "frozen" and one team is randomly selected from it to compete against pop1. In one generation, each team in pop1 plays 30 games against the team from pop2. Then the teams in pop1 are ranked according to their fitness scores. After ranking, the bottom half of the teams in pop1 are removed and the remaining half is duplicated. These duplicates are then mutated by adding random Gaussian noise to the weighting functions. This allows the learning algorithm to remove poor performing teams and search out better solutions similar to the successful ones. After mutations, one generation of evolution is complete and the cycle repeats. After 100 generations, the evolution switches to the other population. Pop1 is frozen and all the teams within pop2 play 30 games each against the best team from pop1.

The fitness score for this scenario is carefully formulated. Unlike previous swarms where a single value could be tracked [6], such as time survived, this swarm has multiple goals. A successful swarm should be able to guard its own base long enough to find and destroy the opponent's base, while at the same time keeping as many agents alive as possible. A swarm that attacks its opponents base can receive up to 50 fitness points based on how much damage it inflicts on its enemy's base, along with up to 50 more points based on how much strength its own base has remaining at the end of the simulation. If a swarm successfully destroys its opponent's base, a bonus of up to 50 points can be awarded depending on the number of surviving agents in the victor's swarm. If a team does not attack its opponent, it receives one point if it at least finds its opponent's base, and zero points if it fails in its search.

Figure 5. Evolution Results for the Point Attack & Point Defense Swarms

C. Results

The fitness scores over the course of the evolution process are shown in Figure 5. Only the fitness scores of the "active" population being evolved are shown. The populations take turns being evolved against the other every 100 generations. These switches occur at the discontinuities on the graph. Initially, pop1 is the active population for 50 generations and achieves excellent scores. This is not that impressive, however, considering their opponent is a just a team with randomized weighting functions. At generation 50, pop1 is "frozen" and pop2 is evolved. Since pop1 is much more highly evolved than pop2, pop2 has a difficult time competing against pop1 and receives poor fitness scores. This pattern repeats for almost the next 1000 generations, until pop2 finally catches up to pop1 and they both become equally matched opponents. After approximately 1200 generations, the algorithm is able to optimize the parameters of the swarm to maximize its ability to both defend its base and attack the enemy base.

For a highly evolved swarm, when the simulation started, approximately two thirds of the swarm switches into an aggressive mode, with the rest of the agents acting as defenders. The defenders form a small mob around their friendly base to fend off enemy attacks. A larger number of agents are required for the offensive task due to the fact that spreading out across the map to find the enemy base takes more units to complete as opposed to the smaller number needed to defend the base. When opposing agents approached either other, the stronger group of agents usually begins chasing the weaker group, since the outcomes of battles are determined using the relative strengths of the agents. As the simulation progresses, defending agents slowly die off, only to be replaced by nearby agents that determine that they are safe enough to do so. This process allows each swarm's defenders to

guard its base until its attackers find and attack its enemy's base. Eventually, as both sides incur losses, the swarms begin to break down and cease to function as swarms.

The Point Attack & Point Defense swarm simulation demonstrates how techniques found in swarms of ants can be applied to a battle type scenario. By allowing the agents to switch between tasks during the simulation, the swarms are able to perform well for a longer period of type and can adapt to their current circumstances. The optimized rules applied as a set of weighting functions and threshold functions, while simple, were able to result in a flexible and robust emergent behavior which allows the swarm to complete two separate objectives. Coevolution is shown to be an effective means of optimizing the parameters for the scenario.

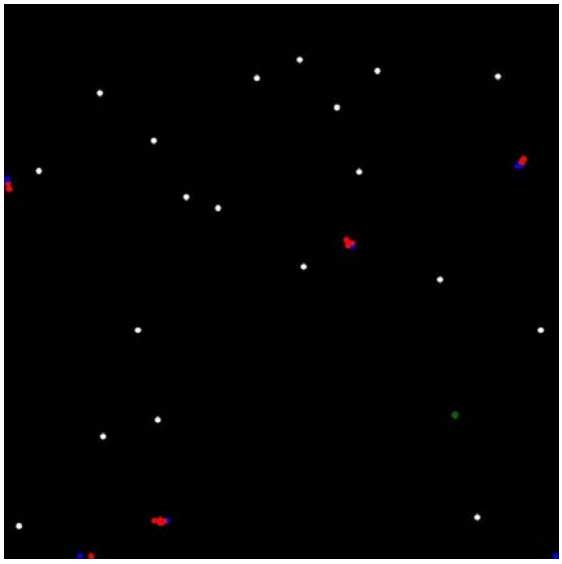

Figure 6. Screenshot of the Search & Destroy swarm. White dots are explorers. Blue dots are recruiters, and Red dots are soldiers. The enemy target is represented by the green dot.

IV. Search & Destroy

A. Simulation

The second simulation for the swarms to learn is a Search & Destroy mission, shown in Figure 6, wherein a swarm is tasked with finding and eventually destroying an enemy target. Unlike the previous example, this enemy is allowed to fight back. In order to encourage the emergent behavior of recruitment, the enemy is able to withstand attacks of less than 3 units. The enemy is also able to inflict a random amount of damage, ranging from zero to ten, against attacking agents which are initialized with a strength value of 100. When an agent's strength drops to zero, it is removed from the field of play. This process allows for the enemy to survive attacks of less than three agents and destroy the failing agents. The simulation continues until either the target is destroyed, the swarm dies out or the time limit expires.

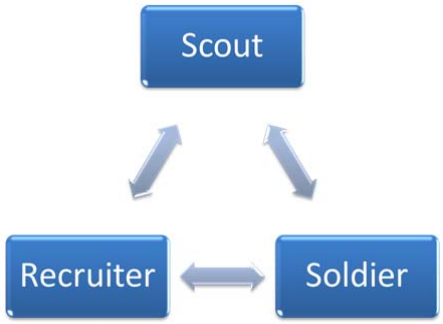

Figure 7. The three possible states in the Search & Destroy swarm.

Agents in this swarm have the ability to take on one of three states: scouts, recruiters and soldiers. In addition to deciding when to switch states, the agents also need the ability to recruit units in order to form attacking groups. To draw another comparison to ant colonies, when ants have trouble moving large objects, their first method of recruitment is to release a large amount of pheromone within a local area. If that does not attract enough ants, the ant will return to the anthill leaving a trail of pheromone behind. Since the pheromone trail method does not work well with all applications, we decided to implement only the first method. In the Point Attack & Point Defense scenario, our goal was to explore the evolution of the agents' state switching behavior. In this simulation, however, our focus was les on how the agents switch as it was on allowing the swarms to evolved recruitment methods. In order to simplify the evolution process, the weighting functions that define the agents' behavior are evolved, but the state switching itself follows a set of pre-defined rules.

All agents are initialized as scouts. When a scout finds the enemy, it becomes a recruiter. When a scout finds a recruiter, it switches to the soldier task. A soldier will remain in that state unless it gets separated from its recruiter, in which case it reverts back to acting as a scout. Additionally, if a recruiter finds another recruiter, the recruiter with the lowest remaining strength becomes a soldier and treats the other agent as its recruiter. This rule was added to the simulation to prevent large numbers of recruiters in the swarm searching for scouts, when instead they could simply form up with each other in order to more quickly accomplish their objective. Each state contains its own set of weighting functions. So when an agent is said to have "switched states" it is actually switching the set of weighting functions that govern its behavior.

As with the previous example, the movement of the agents is controlled by sensors and their corresponding weighting functions. Sensors used include nearest agent of the same state, nearest agent of a different state and the enemy target. Additionally, soldiers are able to sense their nearest recruiter and recruiters are allowed to count the number of soldiers within sensor range. Another sensor is introduced to this simulation: a center sensor. In the Point Attack & Point Defense swarms and in much of the previous swarm work with our group, most of the interesting behavior occurs at the boundaries of the playing grid. The rectangular grid with rigid boundaries is normally used for simplicity, but we want to replace these hard boundaries with a softer, circular boundary. To

accomplish this, we remove the boundary and replace it with a center sensor. Agents are allowed to travel "off the map" as far as they want, but agents that are too far away at the end of the simulation are considered lost. The center sensor allows the swarm to keep agents from wandering off by pulling them back in. The sensor is similar to the other weighting functions, but is defined by only two values: the distance from the center at which the sensor is turned on and the strength of the agent's attraction back to the center. While the initial reason for this sensor was to contain a swarm in a continuous field of play, the sensor ended up being used by the swarms to improve the effectiveness of their search, which will be explained later.

It is important to mention that, while this scenario is designed with the goal in mind to develop recruitment techniques, the swarm is still given flexibility to find an optimal solution. Instead of hard-coding the behaviors into the program, the framework for the behavior is provided, and the swarm is allowed to adjust various weighting functions and thresholds to refine this behavior. In studying swarm intelligence, we are often searching for both interesting behavior and useful behavior. The "bullies v dweebs" scenario [13] is an example of some very interesting work, where the emergent behavior was unexpected. However, the simulation does not directly translate to an application, which is the focus of this research. While the simulations described in this paper are designed with the goal of providing useful techniques, the flexibility of the swarm was maintained by allowing the weighting functions to be evolved.

B. Evolution

To evolve this swarm, coevolution cannot be used since there is only one swarm involved. Instead, a single population of teams is generated and optimized by removing poor performing teams and replacing them with mutated copies of the remaining teams. In formulating the fitness function for this simulation, we need to think about how to steer the evolution towards our goal, while at the same time allowing the swarm to find unexpected results. For the fitness function, if the swarm at least found the target, it receives one point. A swarm can also receive up to 100 points depending on how much damage it inflicted on the target. The total distance travelled by the swarm is tracked through each simulation. If the target is successfully destroyed, up to 50 points are awarded based on how much distance the total swarm travels compared to the maximum possible distance travelled. Less movement was awarded more points to promote efficiency and conservation of movement translating into real-world energy and fuel savings. Finally, the total fitness score is adjusted by multiplying by the percentage of swarm remaining.

During one generation of evolution, each team is simulated twenty five times and the program records the percentage of games in which the swarm successfully destroys the target, the percentage of games in which the swarm at least finds the target and the average fitness score. The teams are compared against each other in pairs in order to rank the teams. Each team is matched up with ten other, randomly selected teams. For each match up, the teams are first compared using the percentage of successful simulations. The team with the higher percentage "wins" and is awarded one point. If the teams tie in this test, the next criteria is the percentage of games with a successful search. A point is again awarded to the team with the higher percentage. In the case of another tie, the average fitness score is used to determine who received the point. Teams are then ranked according to their point totals. At this point, the lower half of the teams are removed from consideration and replaced with mutated copies of the better half. The point system allows the "natural selection" process to be more forgiving. A team that achieves worse scores temporarily is able to survive longer and possibly discover a better strategy later on in the process as opposed to being removed immediately after they fall into the bottom half of teams in the population.

C. Results

After evolution, the swarm learns to successfully find and destroy the enemy target within the time limit. Scouts are repelled from each other, which causes them to spread out across the map. When a scout wanders off too far from the center, it is kept alive by gently being pulled back in. After a scout finds the enemy target and becomes a recruiter, it stays away from the target until it senses it has a large enough group of soldiers around it (at least two, for a total group of three). At this point, the recruiter returns to the target, attacks it and usually destroys it. An interesting result is that when a scout switches to a recruiter, it begins searching for soldiers within a circle of a smaller radius than before the switch. Recruitment occurs when a recruiter comes within range of either a scout or another recruiter, so by staying in a smaller search area, the recruiters are able to increase their chances of finding other recruiters who are also staying in the same area. This behavior emerges from the use of the center sensor, which is "turned on" at a smaller radius when an agent is in the recruiter state. The use of the center sensor for recruitment was unexpected and is another example of why it was important in these simulations to allow room for the evolution process to discover the optimum solution.

During this process, occasionally the swarm would find a "trivial" solution, that while not what we were looking for, did provide some interesting and unexpected behavior. For instance, originally this simulation was designed to initialize the target at a random distance from the center, ranging from 400-480 units away. The emergent behavior after that variation of the scenario was evolved was the formation of a large group. Instead of spreading out to search for the target, most of the agents formed one big mob, traveled to a distance of roughly 440 units away from center, and simply went in a circle around the center, eventually running into the target and destroying it almost instantly. Since the sensor ranges in this case were 30 units, agents traveling in a circle of radius 440 (with some twiddle added in) were able to find virtually any target hiding within the 400-480 circle. This meant that the search problem was actually a trivial one. With the search task solved by travelling in a circle, the need for recruitment was removed by performing "recruitment" in the beginning. When attacking the target, it makes no difference whether a scout or soldier is attacking, so the evolution converged to a solution where instead of a single recruiter and a handful of soldiers attacking, a large group of scouts formed up and took out the target themselves. This unexpected (and unwanted) behavior is a perfect

example of how we need to be careful in how we design the problem the swarm will be solving. In this case, the search problem was too easy, which made the entire result trivial. Any simulation we design will have an implicit maximum fitness, ranging from trivial to impossible. The search for an interesting, yet solvable problem is one of the more difficult challenges in swarm intelligence.

At one point, we attempted to introduce the concept of memory into the system, with the hopes of improving the effectiveness of the search. Units would be able to remember their previous three locations and, theoretically, would learn to not search the same space twice. Our result, however, was that the agents lost their cohesion, and the final swarm result suffered. Each agent, now having information on its previous locations, did learn to move away and look for new areas, but in the process the agent would effectively ignore all other units nearby. The agents were each searching the space independently instead of cooperating with each other by spreading out. While the fitness scores were lowered by the addition of memory, they could have been improved by evolving a zero weight on the trails. Zeroing out the trail weights, however, did not occur in our evolution due to the fact that the odds of all of the necessary weights reaching zero simultaneously was extremely low. In effect, the evolution would always converge to a local maximum of the fitness function that was too far away from the global maximum. Based on these results, we decided to refrain from implementing a temporary memory for each agent.

The Search & Destroy simulation, while simpler than the Point Attack & Point Defense swarms, demonstrated the use of both state-switching and recruitment. The techniques discovered in these scenarios lays the groundwork for more complex swarms later on. Also, the evolutionary algorithm using a points system comparison between teams was shown to be a useful method to invert the swarms.

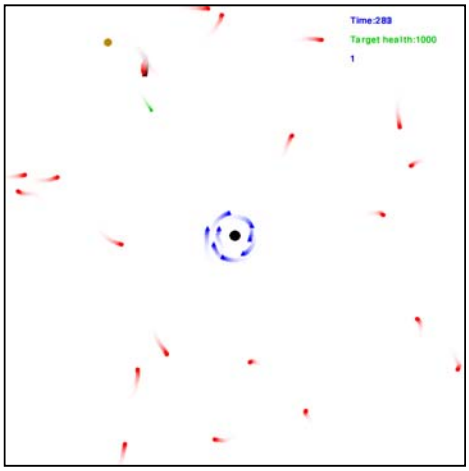

Figure 8. Base Attack. The base is shown in the middle by a black dot. A yellow agent has appeared in the upper left and fired a green projectile toward the base. Agents are colored as follows: red scouts, blue defenders, and maroon recruiters.

V. Base Attack

A. Simulation

Concepts from the first two simulations are combined into a more complex, Base Attack scenario, pictured in Figure 8. The Base Attack swarm has two objectives. First, the swarm needs to defend a central base from incoming projectiles. Agents can detonate themselves to destroy the enemy projectiles. However, these explosions also take out friendly units nearby. Second, the swarm needs to seek out enemy units that are spawned at the edges of the playing area. These enemy units periodically fire projectiles towards the central base. Enemy units are again stronger and require three agents to detonate nearby within a short period of time. This requirement is added to force the swarms to form groups which would involve learning recruiting techniques. When the friendly base takes too much damage and is destroyed, the simulation ends. A effective swarm in this simulation would be able to defend its base while simultaneously searching for and destroying enemy units. Swarms are scored on how long they survive and how many agents stay within bounds. The solution requires the swarm to allocate its resources between the two tasks and find efficient methods to complete its objectives.

The movement of the agents is again controlled using the input from a variety of sensors. The agents are able to sense their distance from friendly units, enemy units, enemy projectiles, and the base. These distances are fed into weighting functions to determine the resulting movement. The weighting functions are adjustable parameters that represent the rules that the swarm follows. After a solution is found, we can look at the resulting weighting functions to determine which strategies were learned by the swarm. All of the sensors have a limited range, so that agents are only aware of what is happening within a localized area.

Since there are two main objectives the swarm is trying to accomplish, there are two states the agents are allowed to take: attackers and defenders. Defenders are equipped to defend the base by destroying incoming projectiles, while the attackers are capable of searching for the enemy units, forming groups and attacking the enemies in force. Within the attacker task, there are two sub-states: scouts and recruiters.

Similar to the Point Attack & Point Defense swarms, the switching behaviors of the agents is modeled as a threshold function. The threshold functions determine the percent chance that an agent will decide to switch based on the input of an environmental variable. In this case, switching is partially determined by the number of units in the same state versus the units in a different state. Again, the agents are only aware of local information, so the switching sensors have a limited range as well. Each function is defined by a single variable, the threshold. At the threshold value, the output of the function is 50%. If the input is less than that value, then the agent will most likely not take any action. As the value increases above the threshold, the agent will be more inclined to switch states, if the conditions are right. To prevent the swarm from making the mistake of ignoring enemy units, agents are only allowed to switch when there are no enemy units or projectiles nearby. This behavior could be evolved by the

learning algorithm, but would require a more complex switching algorithm, involving the number of both friendly units and enemy units. To simplify the simulation, this rule is hard-coded into the agents' decision making process. While there is a chance that an agent can oscillate between states, that chance is minimal enough to ignore in this simulation.

In addition to sensing the number and state of nearby units, we utilize a "smart" base. While the central base is not a part of the swarm, we allow the base to interact to a small extent with the nearby agents. The base counts up the number of nearby defenders and broadcasts that number to nearby agents. Agents within range are then able to make decisions on whether or not to switch based on the information given by the base. This was necessary because, in some cases, agents would think the base was unguarded when, in fact, it was, but the defenders were out of range on the other side of the base. Agents are able to decide for themselves when the number of defenders is either too large or too small. The base is not actually making any decisions by itself. Instead, it is passively sending the information for the agents to process.

This swarm takes advantage of recruitment techniques developed in the Search & Destroy swarm. Within the attackers' task, there are two sub-states: recruiters and scouts. All attackers are initially scouts. For simplicity, the state switching within the attacker objective is controlled via some preset rules. When a scout finds an enemy, it becomes a recruiter. When defenders or other recruiters find a recruiter and determine that they are in a safe position, they become scouts. The goal is for recruiters to search for other agents until a large enough group surrounds the recruiter so that the enemy unit can be destroyed by the group. Another rule was introduced that allowed units that returned to the enemy's location but could not see the enemy to switch back to scouts and continue searching. In this scenario the enemy may have either drifted away or been destroyed by another group of agents. In either case, the agents should move on instead of getting stuck in a location that may not be important. While the recruitment itself is not an adjustable parameter, the movement of the units within the recruitment sub-states is adjustable. Scouts need to learn to follow recruiters and recruiters need to learn the optimal size of groups needed. In this simulation, three agents are needed to destroy the enemies.

B. Evolution

For the evolution of the swarms' parameters, we looked at a variety of evolutionary strategies [4][5][7][9][10][11]. After some experimentation, we selected a method similar to that used in David Fogel's Blondie24 program. Fogel's program was successful in evolving neural networks that could play checkers [2] or chess [14]. At the beginning of the evolutionary process, a population of teams is generated. Each team contains a set of weighting functions and threshold functions that define the rules followed by the team's swarm. For this experiment, a population of 50 teams was used.

During each round, each team plays a set number of games. After the simulations are completed, each team receives a fitness score that represents how well the swarm performed during the simulations. It is often difficult to determine a fitness function that rewards both good defensive and offensive strategies. In order to encourage the swarms to learn to defend the base as long as possible, points are awarded to the teams based on how long the base survived. This point total is then modified by multiplying the percentage of active units that remain in the playing area at the end of the simulation. This encourages swarms to learn to stay within the playing area without actually setting a hard boundary. The fitness scores are also adjusted by adding bonuses for conservation of movement.

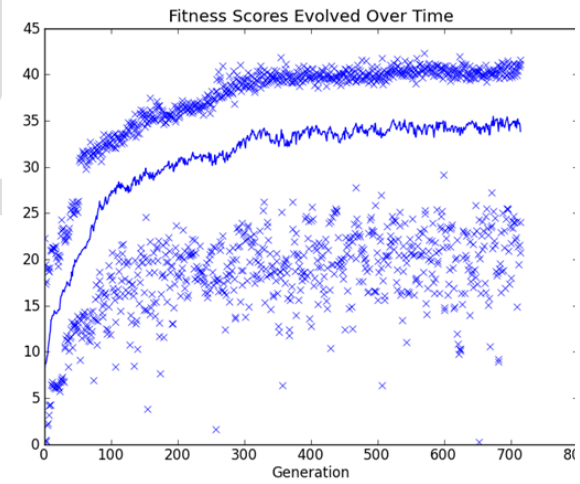

Figure 9. This figure represents the learning process of the evolutionary algorithm by showing how the fitness scores improved over time. The solid line indicates the average fitness score of the population for each generation. The X's represent the maximum and minimum scores in the population. After 300 generations, the algorithm converges to what appears to be a locally optimal solution.

Then, using a lexicographical sorting method, the teams are selected based on the number of games in which they accomplished certain objectives. After the teams are sorted by fitness, they are sorted based on the number of games where they find at least one enemy. This rewards teams that successfully complete the search objective of the attackers. Finally, the teams are sorted based on successfully destroying enemies, which indicates a completion of the second attacker objective, destroy. At this point, the worst 25 teams (half of the overall population) are removed from the evolution process and the best 25 teams are duplicated.

The new 25 teams are mutated by adding random Gaussian noise to the weights that control the swarms' behavior. This process allows the evolutionary algorithm to remove poor solutions and keep successful solutions, while constantly searching for new and improved strategies that are both similar to previous good solutions and different enough that the search is considering new strategies. The mutation step size is an important parameter in the evolutionary program. If the step size is too small, then the program will not be able to effectively search through the entire search space. On the other hand, if the step size is too large, then the search will not be able to converge to a solution. In order to prevent the search from converging too quickly, a minimum step size was used. The minimum step size was calculated by first calculating the average step size for each weight over all of

the teams tested. After a list of the average step sizes was calculated, the minimum step size was found by selecting the median of the average step sizes using the method described by Liang et al. [8].

In Pseudocode:
for i in number of generations:
Simulate each team 50 times
Rank by fitness scores
Rank by # of times enemy found
Rank by # of enemies killed
Remove bottom 25 teams
Duplicate top 25 teams
Mutate duplicates

C. Results

After the evolutionary algorithm was run for several hundred generations of teams, the resulting strategies allowed the swarms to perform well in both the defense of the base and the search for and destruction of enemy units. The improvement of the fitness scores over the course of the evolution process is shown in Figure 9. The learning algorithm allows the swarms' fitness scores to increase over time, before leveling out at a maximum value given the parameters of the simulation.

One of the basic behaviors learned was the division of labor. A swarm was able to divide itself up into two groups, with roughly two thirds going into attack mode and the remaining acting as defenders. This emergent behavior was intuitive given that attackers had more of a search area to cover, while only a small amount of defenders were needed to guard the base. After the initial division of labor, the swarm was able to dynamically shift its resources autonomously. As defenders are depleted through either enemy projectiles or recruitment, they are replenished by nearby attackers that switch states when they determine the number of defenders is too small. Figures 10 and 11 demonstrate the dynamic state switching behaviors learned by the swarms.

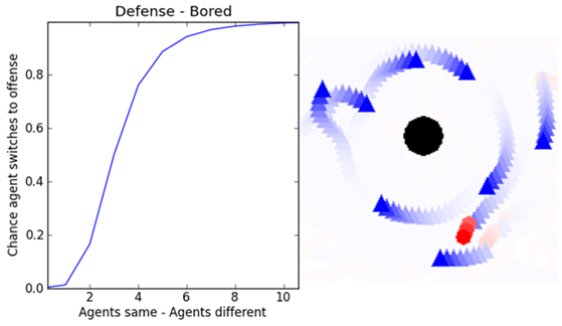

Figure 10. The threshold function shown here demonstrates role saturation, which occurs when agents switch states after deciding there are too many units working on their task. First, the agent counts up the number of nearby agents working on its task and those working on a different task. This is fed into the threshold function, which determines the chance that the agent will switch. In this case, if the difference between nearby defenders and attackers is 3, then the agent has a 50% chance of switching to offense. If the number of defenders compared to attackers is large, the agent will most likely switch, and vice versa. In this image, a blue defending agent has decided that there are too many defenders around it and chooses to switch states to become a red scout.

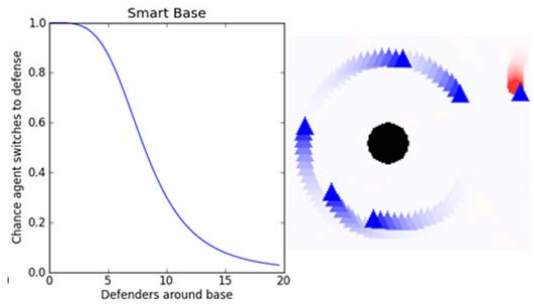

Figure 11. This function represents the way the agents process the information broadcast from the base. The base will broadcast the number of defenders around it and the agent has the option of switching to a defensive mode if it decides there are not enough defenders around it. In this case, the swarm will attempt to keep at least 8 or so defenders around the base. If the number of defenders is less than that, then there is a role deficiency and nearby attackers will most likely switch to a defensive mode. Here, the base has broadcast that there are 6 defenders around it. A red scout has heard the message and decided that 6 defenders is not enough, so it chooses to switch states to become a blue defender.

The attackers learned to spread out both from the base and from each other. The scouts also learned an optimal distance at which to turn on their center sensor to allow them to both remain in the playing area and search as much of the map as possible. This attacker behavior is shown in Figures 12 and 13.

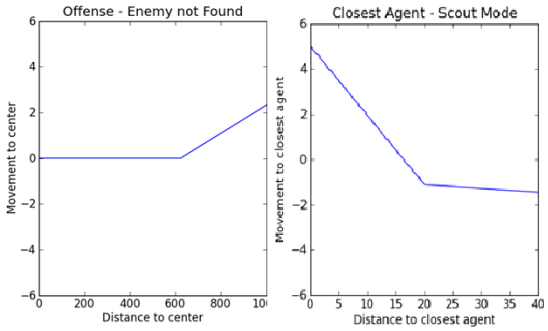

Figure 12. This weighting function shows how the swarm has learned to stay within the boundaries without being explicitly told how to. Any units that are greater than approximately 850 units away from the base are considered lost. Teams can achieve higher scores by keeping a large percentage of their agents within bounds. Also, the playing area is 1200 by 1200 units and the enemies are spawned at a radius of 600 units from the base. This team has learned that if the agents turn back to the base after they are a distance of 620 units away, they will remain within bounds while still being able to find all enemy units. This graph shows how scouts react to seeing other units. When a scout sees another agent, it will be repelled from it. This allows the scouts to spread out and cover as much area as possible. Additionally, the graph is positive for small values. While the scouts are initially repelled from other agents, they will also be attracted to recruiters through the use of a separate weighting function. When the scout is pulled in to a close distance from the recruiter, the positive value from this weighting function allows the scouts to follow the recruiters more closely.

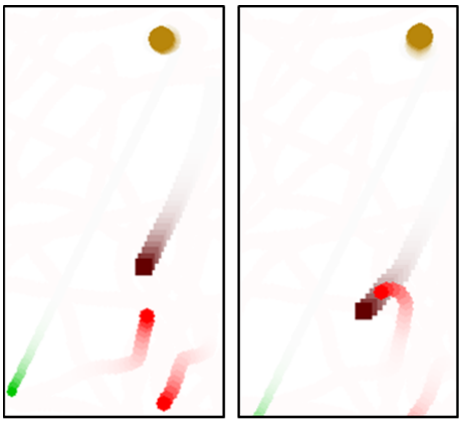

Figure 13. In this screenshot, a red scout approaches a maroon recruiter. While the scouts are inclined to stay away from other agents when they are relatively far apart, their attraction to recruiter overwhelms the initial repelling force. When the recruitment sensor brings the red scout close enough to the recruiter, the scout's attraction to other units at close distances kicks in and the scout will tightly follow the recruiter until the enemy is eventually destroyed. A green projectile fired from the enemy is also pictured.

The defenders also learned to surround the base while maintaining a set distance from each other. They learned to keep their distance because detonations to destroy enemy projectiles could destroy friendly units if they were too close. Figures 14 and 15 show the defensive strategies used by the swarms.

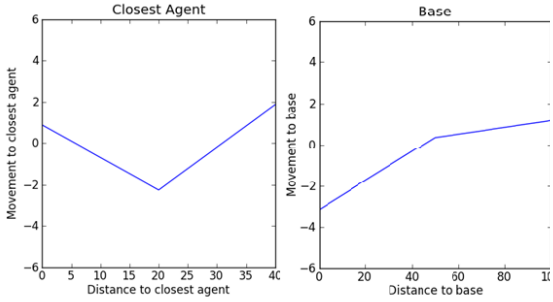

Figure 14. This weighting function shows how defensive agents will stay approximately 31 units away from each other. The zero crossing with a positive slope creates a "sweet spot" that the agent is inclined to stay in. Note, the function does become positive for small distances, but since the function is negative for values from 6 to 31, the agents should never get close enough to each other for that to matter. This function represents the rule that tells the defensive agents how far to stay away from the base. The agents will attempt to remain about 45 units away from the base.

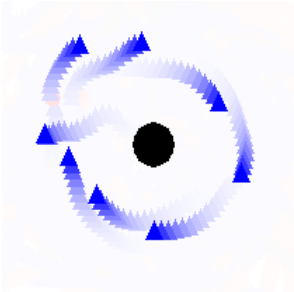

Figure 15. The defending agents learned to surround the base and travel in a circular pattern in order to intercept as many enemy projectiles as possible. They are also keeping a set distance away from each other so that is one detonates, it doesn't take out friendly units.

One of the more unexpected results came from the optimization of the center sensor. The goal was for the swarm to learn to stay within the boundaries of the playing area. An interesting emergent behavior was that the recruiters' center sensor turned on at a very small value. This caused all recruiters to be drawn back to a tight radius around the base, which resulted in an effective recruitment strategy as there is always a group of agents close to the base. These recruiting techniques are shown in Figures 16 and 17.

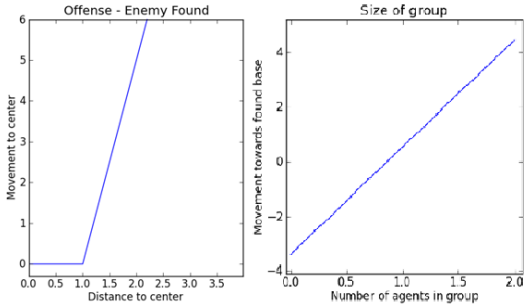

Figure 16. The first function shows the center sensor for recruiters. Note the scale for the x-axis. Any recruiters that are more than 1 unit away from the base will be inclined to return to the base. In other words, all recruiters return to the base. It is easy to understand why this unexpected strategy developed because there are (or should always be) agents acting as defenders near the base that can be recruited to join recruiters' groups. This second graph demonstrates another part of the recruitment method learned. It represents how the agent moves with respect to the found enemy based on the number of friendly agents around it. If there are no friendly units around, the agent is repelled from the enemy; it is not strong enough. If there is one unit around the recruiter, then it still does not return to the enemy. Only when there are at least two friendly agents nearby does the recruiter return to the enemy. At this point, the group is at least three agents strong and able to destroy an enemy unit.

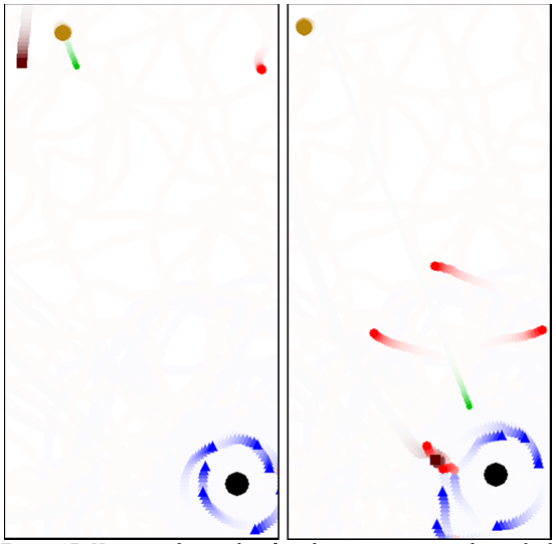

Figure 17. Here, a red scout has found an enemy unit and switched states to become a maroon recruiter. The recruiter is headed back toward the base in order to recruit other agents to form a group large enough to take out the enemy. This image shows the recruiter being joined by a third agent, which was formerly a blue defender. Since the recruiter senses that its group is at least three agents and is big enough to destroy the enemy, the recruiter turns and begins leading the group to the enemy unit to attack it. After the enemy is destroyed, any remaining agents from the group will continue searching in a scout mode.

One of the benefits of swarm intelligence is graceful degradation of the swarm's performance. As the simulation progresses, the swarm will incur losses. However, by dynamically shifting its resources, the swarm is able to maintain both tasks, defending the base while still searching for enemy units. It is only when the swarm loses a large percentage of its population that the swarm begins to break down and is no longer able to successfully work on both objectives. The swarms in this project were evolved with an initial population size of 40 units. This number allowed the group of units to be large enough to be considered a swarm while still being small enough to encourage unique, emergent behavior. The concept of how large a swarm needs to be in order to be considered a swarm is a fuzzy one and often depends on the application. The question of how size affects a swarm's performance will be explored further in future work.

VI. Conclusion

One of the advantages of swarm intelligence is a swarm's ability to autonomously reorganize itself in a dynamic environment. In our work, we have used techniques found in nature to allow swarms to manifest this behavior in simulations where the swarm is required to perform well in two objectives. In the Point Attack & Point Defense swarms, agents have to balance themselves between both defending their base and finding and attacking their enemy's base. Swarms in the Search & Destroy simulation have to use recruitment methods to form groups to destroy a large opponent. And finally, these scenarios are combined in the Base Attack swarm, in which a single swarm has to complete defensive and offensive objectives, using both threshold functions and recruitment techniques. By using an evolutionary learning algorithm, the weighting functions that defined the swarms' behavior in each of the three simulations are optimized to maximize the swarms' fitness scores.

We believe that these concepts can be expanded upon in future work. One topic to consider is the effect of size on a swarm's performance. For the purposes of these simulations, a population size of 40 was chosen because it is small enough to be feasible in a real-world application but also large enough to demonstrate swarm characteristics. A more in depth exploration of the effects of population size could provide more insight as to when a large group of agents begins functioning as a swarm.

This paper has demonstrated the application of a multi-state swarm that was able to use state-switching capabilities to adapt to a dynamically changing environment. While previous work has shown swarm intelligence as a viable solution to single objective missions, we have expanded these swarm techniques to accomplish multiple objectives using threshold functions to control the switching between states. The emergent behaviors of the swarms are robust and allow the swarm to continue achieving its objectives until a large percentage of its population is lost.

Acknowledgment

Special thanks to Baylor University, the Pennsylvania State University Applied Research Laboratory, and especially the Office of Naval Research's University Laboratory Initiative for funding for this effort.

References

[1] E. Bonabeau et al, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford, NY: Oxford University Press, 1999.

[2] D. Fogel, *Blondie24*. San Francisco, CA: Morgan Kaufmann Publishers, 2002.

[3] I. Gravagne and R. Marks II, "Emergent Behaviors of Protector, Refugee, and Aggressor Swarms,"*IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 37, no. 2, pp.471-476, Apr, 2007.

[4] Z. Yuan, "Continuous Optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms," ANTS 2010, pp. 203-214, 2010.

[5] M. Clerc and J. Kennedy, "The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58-73, Feb, 2002.

[6] W. Ewert, R.J. Marks II, B.B. Thompson & Albert Yu, "Evolutionary Inversion of Swarm Emergence Using Disjunctive Combs Control," IEEE Transactions on Systems, Man & Cybernetics, (prepint available at IEEE Xplore February 1, 2013.)

[7] D. Cvetkovic and I. Parmee, "Evolutionary Design and Multi-ojbective Optimisation," Plymouth Engineering Design Centre, University of Plymouth. Drake Circus, Plymouth PL4 8AA, U.K.

[8] K. Liang et al, "Dynamic Control of Adaptive Parameters in Evolutionary Programming," Computational Intelligence Group, School of Computer Science. University College, The University of New South Wales. Australian Defence Force Academy, Canberra. ACT, Australia 2600.

[9] C. Fonseca and P. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," Dept. Automatic Control and Systems Eng. University of Sheffield, Sheffield S1 4DU. U.K. July, 1994.

[10] F. Kursawe, "A Variant of Evolution Strategies for Vector Optimization," University of Dortmund, Department of Computer Science XI, D 44221 Dortmund, Germany.

[11] S. Carlson, "A General Method for Handling Constraints in Genetic Algorithms," University of Virginia, Charlottesville, VA.

[12] Jon Roach, R.J. Marks II & Benjamin B. Thompson, "Tactical Task Allocation and Resource Management in Non-stationary Swarm Dynamics," IJCNN 2013

[13] Jon Roach, Winston Ewert, Robert J. Marks II and Benjamin B. Thompson, "Unexpected Emergent Behaviors From Elementary Swarms," Proceedings of the 2013 IEEE 45th Southeastern Symposium on Systems Theory (SSST), Baylor University, March 11, 2013

[14] D. Fogel, et al., "A self-learning evolutionary chess program," Proceedings of the IEEE, vol. 92, no. 12, pp.1947,1954, Dec 2004