ABSTRACT

Emergent Behaviors of Multi-Objective Swarms with Applications
in a Dynamic Underwater Environment

Jon H. Roach, M.S.E.C.E

Mentor: Robert J. Marks II, Ph.D.

The allocation of resources between tasks within a swarm of agents can be difficult without a centralized controller. This problem is prevalent when designing a swarm of Autonomous Underwater Vehicles, in which underwater communication becomes challenging and a centralized controller cannot be used. In this thesis, a disjunctive fuzzy control system is used to solve the problem of resource management. Multi-objective, multi-state swarms are evolved with an offline learning algorithm to adapt to dynamic scenarios. Some of the emergent behaviors developed through the evolutionary algorithm are state-switching and recruitment techniques. In addition, the adaptability of swarms is tested by removing sensors from the system and re-evolving the swarm to allow it to compensate for its sensor loss. The concepts of a multi-objective, multi-state swarm are also applied to an underwater minefield mapping scenario, which is used to test the robustness of the swarm with respect to swarm size.

Emergent Behaviors of Multi-Objective Swarms with Applications
in a Dynamic Underwater Environment

by

Jon H. Roach, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of

Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____

Robert J. Marks II, Ph.D., Chairperson

_____

Benjamin B. Thompson, Ph.D.

_____

John M. Davis, Ph.D.

_____

Charles P. Baylis II, Ph.D.

Accepted by the Graduate School

August 2013

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First of all, I would like to thank Dr. Robert J. Marks II and Dr. Benjamin B. Thompson for all their guidance, support, and for giving me the opportunity to work with them the past two years. Their experience and advice has been invaluable. I would also like to thank Maria Medeiros, the Office of Naval Research (ONR) and the University-Laboratory Initiative (ULI) program for their sponsorship. The program has been a huge blessing to me and I have truly enjoyed the experience of working in a naval research lab. In addition, I am thankful to all of my classmates at Baylor, especially Dr. Winston Ewert who provided much needed insight into Python programming. Last but not least, I am grateful to my friends and family for their support and encouragement over the past couple years. I have enjoyed researching the fascinating field of swarm intelligence and this thesis would not have been possible without all of your support. Thank you.

CHAPTER ONE

Introduction

Ants are simple. If a person watches an ant going about its business, there doesn't seem to be a lot going on. An ant might be carrying around a piece of dirt or a twig or maybe it's just searching for food. On its own, the ant is not very impressive and is often only following some basic instincts: look for food, find food, return to hill, and repeat. However, a large group of ants working together can build a complex anthill. And the colony doesn't just build the mound visible above ground, but a complex series of tunnels and chambers that allows the ant colony to be well fed, protected from dangers (except for the occasional human foot), with the opportunity to grow. How can a group of simple ants work together to achieve something so complex without a central source of knowledge to control its behavior? The answer can be found in swarm intelligence.

In swarm intelligence, we study how simple rules followed by simple agents impact the emergent behavior of a group. In a swarm, agents are not aware of global information, only what they can sense nearby and make decisions on based on their limited range sensors. However, through coordination, the agents are able to work together to accomplish a specified objective. This allows the agents to be simpler and possibly less expensive than other control algorithms. Swarm techniques are particularly useful in scenarios where intra-swarm communication is difficult, or even impossible. Other benefits of swarms include fault tolerance, scalability and adaptability.

1.1 Statement of Purpose

A possible application of our research is in the field of autonomous underwater vehicles, or AUVs (not to be confused with UAVs, unmanned aerial vehicles). For a UAV, communication is fairly simple and the drone is usually controlled by a single human pilot via radio. However, controlling an underwater drone is more challenging due to the fact that radio waves do not travel well in water. Since communication is difficult underwater, using a central controller for a swarm of AUVs may be impossible to implement. Our techniques, such as disjunctive control and threshold functions, remove the need of a central controller and allow AUVs to operate for long periods of time without surfacing for communication.

1.2 Objectives and Contribution

With this application in mind, multiple simulations are developed to model a simplified version of a swarm of AUVs in a real-world scenario. Example scenarios include point-defense, search and destroy, and minefield mapping. In addition to the removal of a central controller, another benefit of swarm intelligence is that the emergent behavior of a swarm can often be surprising and unexpected. In order to preserve the element of the unexpected, the swarms are evolved using an off-line learning algorithm. Instead of telling the swarm exactly what to do, possibly resulting in a sub-optimal strategy, the rules controlling the agents' behavior are evolved parameters. This process allows us to discover the set of rules that results in the optimal emergent behavior of the swarm for a given scenario.

Previous work by Albert Yu [5] and Winston Ewert [2] has studied several basic swarms in which every agent in the swarm is identical and follows the same set of rules.

Our goal in this research is to expand upon these previous simulations to study more complex, multi-state swarms. In a multi-state swarm, agents are able to take on one of multiple states, allowing for greater flexibility and robustness of the swarm's emergent behavior. Having multiple states also allows the swarm to perform different tasks simultaneously, which could be directly applied to a real-world scenario.

Energy conservation is another topic that we have explored. In previous swarm research, efficiency has been incorporated by maximizing the number of surviving agents in the evolution's fitness function. In addition to this, we use the fitness function to minimize the total distance traveled by the swarm, reducing theoretical fuel costs and wasted movement. The concept of battery life is a similar point of interest. In our minefield simulation, agents have a limited battery life, with a portion depleted both when active and when stationary. Agents must return to a home base to recharge periodically. This addition moves our theoretical simulations closer to a real-world application. Adaptability is also one of the major benefits of evolving a swarm. In our research, we have investigated the effects of removing one of the agents' sensors. The result demonstrates the robustness of the swarm inversion procedure through the swarm's ability to compensate for this sensor loss by evolving a different emergent behavior.

1.3 Organization of this Thesis

This thesis describes four different swarm scenarios that were developed during the course of the investigation. Chapter two contains a literature review. In Chapter three, we introduce two basic, multi-objective swarm scenarios: a competitive Point-Attack and Point-Defense game and a single-swarm Search and Destroy mission. Each, while simple, demonstrates emergent behaviors including dynamic state-switching and

recruitment techniques. A third simulation is also discussed, which combines the two previous swarms into a more complex, Base Attack scenario. Successfully evolved swarms running this simulation are able to both defend a friendly base, patrol searching for enemy units and form into groups as needed to destroy enemies. Chapter four tackles the question of what happens when a sensor is removed from the swarm. The evolution process actually allows the swarms to adapt their strategies to compensate for the sensor reduction, often resulting in new and interesting emergent behaviors. In Chapter five we demonstrate how a swarm can be applied to map out a group of underwater mines. In addition, this chapter explores the question of how many agents constitute a swarm. Chapter six contains a summary of the results of this research.

CHAPTER TWO

Literature Review

2.1 Swarms in Nature

In the book, *Swarm Intelligence: From Natural to Artificial Systems* [1], E. Bonabeau *et al.* explore the many appearances of swarm intelligence in nature, particularly in groups of insects. This book also shows how much of this swarm behavior can be modeled as a group of simple rules. A good example of this is the method that ants use to forage for food. Ants wander around searching for food. When they find food, they return to their colony, leaving a trail of pheromone behind. This trail is used to communicate the location of the food source to the rest of the colony. Other ants are able to follow the trail to the food source and begin laying down their own trail. Since the pheromone evaporates over time, if there are multiple paths taken by the ants, the shorter path will have more pheromone on it. Eventually more and more ants will take the shorter path and the longer path will disappear altogether. Using this method, the ants' path converges to the shortest possible distance between the food source and the anthill. This behavior has been modeled and used in Ant Colony Optimization (ACO). This search algorithm, based on the few simple rules that ants follow foraging for food, is used in many applications, including the traveling salesman problem [9].

Ants demonstrate other swarm-like behaviors as well. In fact, before laying a trail of pheromones between the food and the colony, an ant will sometimes release another pheromone. Nearby ants will then come and try to help the first ant move the piece of food to the colony (if it is small enough), instead of having the entire colony march out to

the food. The ants are then able to position themselves at exactly the right spots in order to push or pull it as far as needed. If the piece of food turns out to be too large or heavy to move, the ants then break off a smaller piece and return to the hill, leaving a trail of pheromone behind. This occurs without an external force directing their movements.

Recruitment techniques are also seen in bees. Upon returning from a successful search for pollen, a bee performs a dance to communicate the location of the pollen to the rest of the hive. While the goal of this research is not to replicate biological systems, the fascinating results from swarms in nature can be used as inspiration for possible solutions to real world problems.

2.2 Disjunctive Combs Control

Disjunctive control has been shown to be a viable means of coordinating a swarm of agents without a central controller or a complex decision making algorithm [2]. With the standard conjunctive Mamdani control there is a rule defined for every possible combination of inputs. Combs control, on the other hand, defines a set of fuzzy rules for each output. The output when using conjunctive control can be thought of as a series of inputs "anded" together (if A and B and C, then D), while disjunctive control "ors" them together (if A or B or C, then E). This greatly reduces the dimensionality of the rules needed to define a set of outputs and increases the robustness of the system. If one input is removed, conjunctive control fails, while disjunctive control is able to succeed by using other inputs to find the output. Combs control is realized in swarm modeling through the use of weighting functions. A set of fuzzy rules can be represented by a simple, piecewise linear function. By attaching one of these actuator functions to each agent's sensor, a rule is defined that the agent will follow. If that sensor is removed, the

swarm does not fail completely and can be taught how to compensate for that sensor loss by relying more on its remaining sensors, as we will see in Chapter five.

2.3 Elementary Swarms

Interesting and unexpected emergent behavior can result from even elementary swarms following a single rule [6]. For example, if a large group of people in a room are told to choose two individuals and to move to try and keep themselves between the two people, the result is that the entire group of people will end up squeezed together in small groups, or occasionally, one large group. This emergent behavior is not always obvious or intuitive at first, but often makes sense after the result is found through simulation. Similarly, if you ask individuals in the same group to again choose two people and try and keep one individual between themselves and the other individual, the emergent behavior is the group spreading out to the walls of the room.

Some other examples of basic swarms that have been researched include the wood stacking behavior of termites, the size of a cloud of gnats, and a "bullies versus dweebs" game [3]. Simulated termites are able to stack wood by following just two simple rules: if an agent bumps into a piece of wood, it picks it up, and then puts it down if it bumps into another piece of wood. If all of the agents are pushed in one direction on a modular surface, the emergent behavior of the swarm changes from wood-stacking to tunneling. For the swarm of simulated "gnats", the agents move randomly, except the agent farthest from the center is pulled back in. The result is the entire swarm remaining contained within a circle of fixed radius, regardless of the initialization of the swarm. In the bullies versus dweebs game, a small group of bullies is tasked with hunting down a large group of dweebs. Each bully takes a step towards the closest dweeb and each dweeb takes a

step away from the closest bully. The result of these rules is large number of dweeb "massacres" at the edges of the playing area, where a group of dweebs are stacked up and all destroyed at once. This can be prevented by introducing a small random step to the dweebs' decision making, in addition to the step away from the closest bully. This allows the dweebs to survive for a much longer time and avoid massacres. Conversely, the addition of a random step to the bullies does not improve their strategy. It actually causes them to appear "drunk" and their performance suffers greatly. If the number of bullies is increased, however, the new emergent behavior is a large "mob" of drunken bullies, which is much more effective in surrounding entire groups of dweebs and destroying them. In all of these examples, the interesting behaviors result from a large group of simple agents follow a small set of fuzzy rules.

2.4 Evolutionary Learning Algorithms

The emergent behavior of a swarm, while occasionally unexpected, can easily be determined via simulation. The inverse problem, however, becomes much more challenging. Instead of simulating the numerous possible combinations of weighting functions, an effective method of finding the optimum set of weights is the use of an evolutionary learning algorithm. Coevolution is one type of learning algorithm, in which two populations of possible solutions compete against each other. While initially random solutions, over time poorer solutions are removed and the remaining solutions are copied and mutated. Eventually, a solution emerges that represents an optimum set of weights. David Fogel developed a method of coevolution which could be used to teach a neural network a game. He began with a "simple" game of checkers [8]. In his research, a neural net was initialized with no background knowledge of how to win (or even play)

the game. A large population of possible solutions was tested against each other and scores were based on how many games they won or tied. Over time, the program learned what strategies were resulting in more and more wins and its quality of play improved. After a long period of evolution, the virtual player actually became an expert-ranked checkers player. More recently, Fogel has designed a similar program to develop what is now an expert-ranked virtual chess player [7]. Fogel's work is an excellent example of how a set of weights can be optimized using an offline learning algorithm. Similar to evolving the weights that define the output of the neural network, the actuator functions that govern a swarm's behavior can also be optimized through evolution. Over the years, many variations of evolutionary algorithms (EAs) have been developed [4]. In each case, the members of the evolving population are compared using a single objective, or a combination of objectives through a fitness function. A selection procedure is used to choose which members of the population are allowed to advance to the next round of evolution, and which members should be removed from the process. In some cases, reproduction and recombination are used to generate new members of the population under test. In others, surviving members are simply duplicated. For most every type of EA, however, some form of mutation or crossover is performed in order to continue the search for improved solutions. In some algorithms, as the parameters are mutated, another set of parameters keeps track of the mutations themselves. These self-adaptive parameters are used to determine the next step in the evolutionary process. This procedure was developed by Liang et al [10]. This is helpful in keeping the mutation step sizes large enough to find a better solution far away from a local maximum in the fitness function, while still being small enough to be able to converge to a solution.

2.5 Swarm Inversion

These evolutionary learning algorithms have been used to perform swarm inversion [2]. This has been demonstrated in swarms competing in both a "gang warfare" simulation, where two different groups of agents fight against each other and in another scenario, in which a swarm of rabbits was tasked with escaping to their hole before another swarm of foxes can catch them. In both cases, the rules that the agents follow were inverted using coevolution. Each side in the combat simulation took turns developing strategies to compete against the other team.

In the gang warfare simulations, agents had different strength values. Due to the varying strengths of the agents, one of the more interesting emergent behaviors that developed was the protecting of weaker agents by stronger agents. When the swarms approached each other to do battle, the stronger agents would place themselves at the front of the mob, with the weaker agents at the rear. As the evolution progressed, an orbiting strategy emerged. Agents began circling groups of enemy agents in order to attack the weaker agents at the rear of the formation. Another interesting behavior that emerged was the use of decoys. Sometimes, a small group of agents would go off on their own and distract the opposing force, while the rest of the swarm remained at a distance, safe from their enemy's attacks.

For the foxes versus rabbits scenario, the first behavior that emerged was a simple rush strategy. The rabbits would run to the hole as fast as they could with the foxes chasing them. To counter this, the foxes evolved a "confusion" behavior, where the foxes would remain motionless and the rabbits would sit in the middle of the map, confused. Since none of the rabbits were able to escape before the clock ran out, the

foxes received high fitness scores. This behavior occurred because in the rush strategy, the rabbits were not actually running toward the hole, but away from the foxes. When the foxes did not move, the rabbits would not either, effectively trapping them where they were. After another round of evolution, the rabbits easily countered this behavior. The foxes remained still, but the newly evolved rabbits ignored them and moved to the hole with no problems from the foxes. The foxes then developed a flanking strategy that proved effective at corralling the rabbits before the majority of them could escape to the hole.

2.6 AUV Swarms

In recent work, Albert Yu investigated possible applications of defensive AUVs [5]. In the scenario, a defensive swarm was tasked with defending a VIP (very important person) from an incoming swarm of enemy attackers. A coevolution program was again used to alternatively evolve the attacker and defender swarms. The result was the development of multiple strategies and counter-strategies. One of the behaviors evolved by the defenders was a turtling strategy. Defenders hid very close to the VIP, and refused to engage enemy units. Another behavior that developed was a baiting strategy, where the defenders actually ran away from the VIP, leading overly aggressive attackers away from the VIP as well. Other strategies included a more aggressive hunter method, where defenders pursued attacking agents, and a goalie formation, in which agents swarmed near the VIP, attacking incoming enemies, but not wandering too far off.

To counter the variety of defensive strategies, the attacker swarm also found its own strategies. The simplest was the rush method, where all of the attackers moved in to the VIP as fast as possible. More complex behaviors were learned as well. In some

cases, the attackers spread out from each other in order to slip in between defending agents. Other times, the attackers simply did not engage the defenders, which made the defender's use of bait ineffective. Also, if a defensive strategy was overly aggressive, the attackers would tease the defenders by coming close to the VIP and then moving away, drawing out the defenders and leaving the VIP exposed. For both of these swarms, each agent in the swarm followed the same set of rules. The emergent behavior of the group was dependent on which set of rules the agents were following.

2.7 Motivation

As shown in the previous sections, there has been much research into the emergent behaviors of single state, single objective swarms. Both defensive and aggressive swarms have been evolved using learning algorithms. In real-world applications, however, a swarm may need to perform multiple tasks simultaneously. For example, a group of AUVs might be tasked with guarding a friendly unit and patrolling for enemy units at the same time. Swarm intelligence can be used to maintain the balance between aggressive and defensive behavior in the drones. Also, a scenario might require some form of recruitment technique to be successful. A multi-state swarm allows the distinction between agents working on different tasks, such as recruiting or following. These complex swarms, and their applications, are the primary subject of this research.

CHAPTER THREE

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

3.1 Introduction

An important component of swarm intelligence systems is the 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 section, 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 and attacking enemy targets. Section 2.2 provides a brief background of swarm intelligence. In Section 2.3, we describe a point-attack and 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 2.4 describes a search and destroy mission that a swarm is tasked with. 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 2.5. 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.

3.2 Swarm Intelligence

In a swarm, each unit 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][5] focused on designing swarms that had a single objective. These swarms demonstrated the use of Combs control [2] 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 their opponent. Our goal throughout this research is to expand upon the previous work to more complex swarms that can achieve two or more objectives in a dynamic environment.

3.3 Point-Attack and Point-Defense Swarms

3.3.1 Simulation

The first simulation that we investigate is a point-attack and 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 the enemy base. The game is played in a rectangular, two-dimensional grid, as shown in Figure 1. The edges of the grid are rigid, so when an agent runs into the edge, it bounces off in the opposite direction.

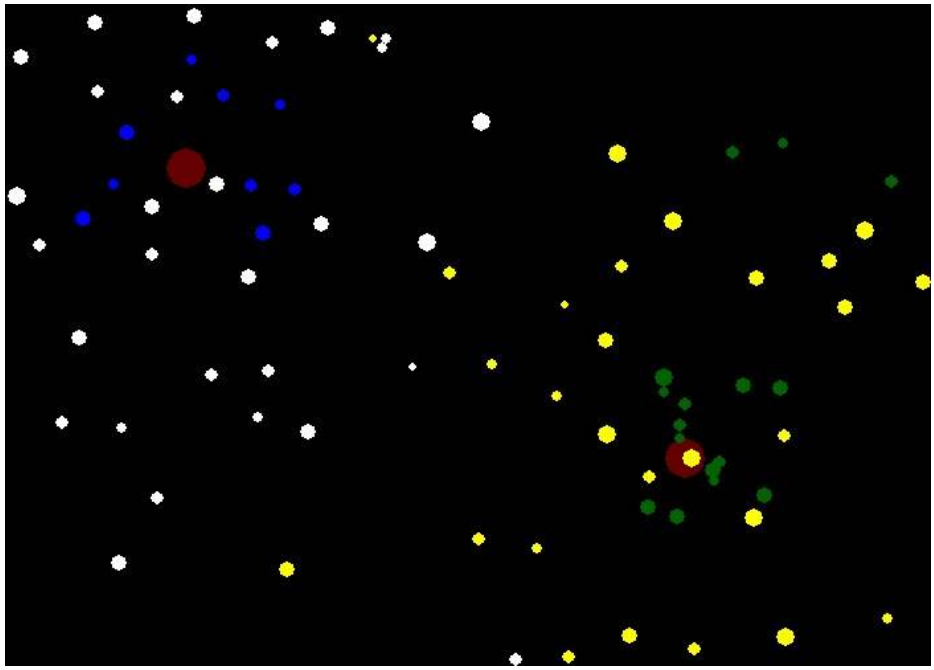

Figure 1. A screenshot from the Point Attack and Point Defense simulation. Swarm one is shown by the blue and white dots, with blue representing defending agents and white, attacking. Swarm two is shown in green and gold for defensive and attacking agents, respectively. Bases are shown in red. The size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

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. This damage reduces the strength of the opponent by that amount. Agents are initialized with a strength value of ten. 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 high 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.

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 Two. The weighting functions come in pairs. One function defines movement toward or away from the object being sensed, while the other 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.

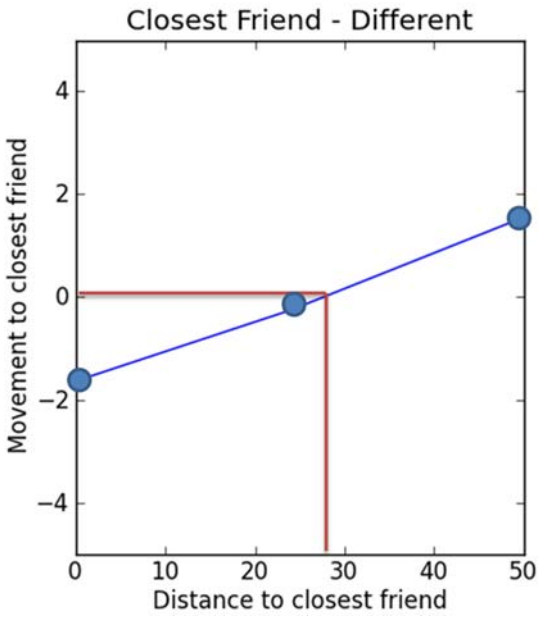

Figure Two. Sample weighting function. This function represents the sensor that finds the distance between an agent and its closest friend of a different state (within range). In this example, an agent within 28 units of the friendly agent is repelled from the agent while an agent at a distance of greater 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

The swarms in the Point Attack and 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 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], similar to the function shown in Figure Three.

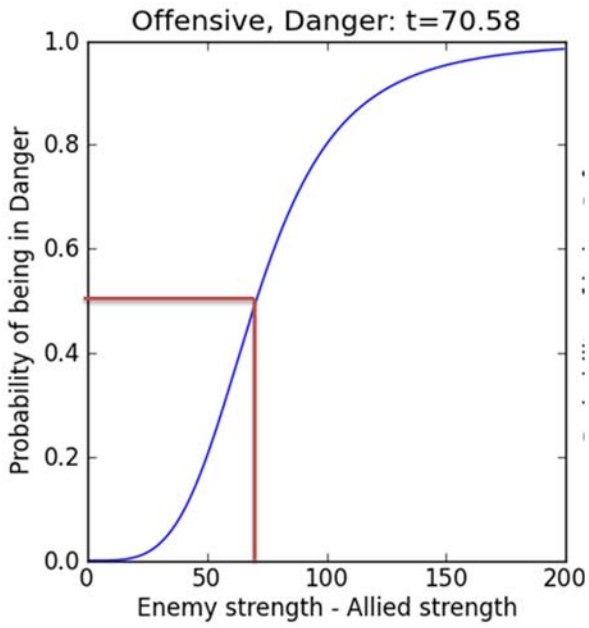

Figure Three. Sample threshold function. This function shows how the difference between enemy strength and allied strength is used to determine whether or not the agent is in danger. In this case, the threshold is 70, so if an agent senses that the difference is less than 70, the agent will most likely 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.

In a similar fashion, agents within the Point Attack and Point Defense swarm switch tasks based on threshold functions attached to sensors that count the strength of nearby agents in different states. If an agent senses a large number of agents around it working on the same task relative to the agent's threshold, it considers itself to be "bored" and 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 is better able to judge when agents are safe or in danger.

3.3.2. Coevolution

The weighting functions are 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 reach a point of equilibrium, where the teams have both maximized their fitness performance and are equally matched. The coevolution process is shown in Figure Four.

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.

The teams in pop1 are then ranked according to their fitness scores. After sorting, 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.

Figure Four. The Coevolution Process.

We carefully formulated the fitness score for this scenario. Unlike previous swarms where a single value could be tracked [2], 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 opponent's 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.

3.3.3 Results

The fitness scores over the course of the evolution process are shown in Figure Five. 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 the 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 starts, 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 approach each 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.

Figure Five. Evolution Results for the Point Attack and Point Defense swarms

The point-attack and 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 time and can adapt to their current circumstances. The optimized rules applied as a set of weighting functions and threshold functions, while simple, are 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.

3.4 Search and Destroy

3.4.1 Simulation

While the point-attack and point-defense swarms produce some interesting results and demonstrate the state switching behavior we are looking for, it does not represent a realistic example of a real-world problem. Drones are currently the subject of intense research [25], but at this point it is unlikely that swarms are needed to oppose enemy swarms. For this reason, we decided to continue our research with a simpler, single swarm scenario. The second simulation for the swarms to learn is a search and destroy mission, shown in Figure Six, 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 fewer than three 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 the target is destroyed, the swarm dies out, or the time limit expires.

Agents in this swarm have the ability to take on one of three states: scouts, recruiters and soldiers, as shown in Figure Six. 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. The pheromone trail method does not work well with our application, so we decided to implement only the first method.

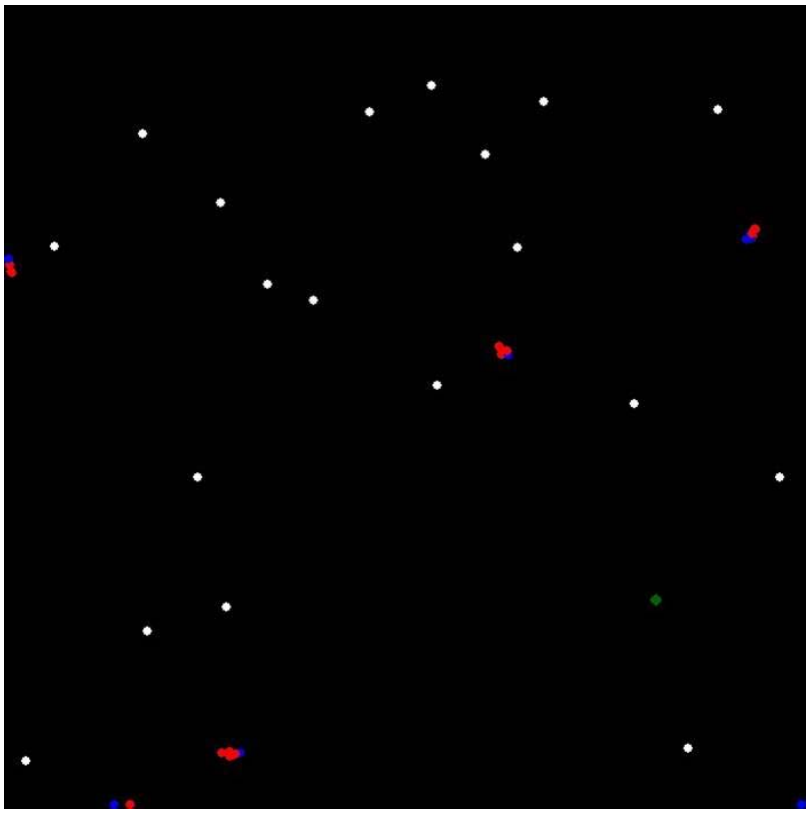

Figure Six. Screenshot of the Search and Destroy swarm. White dots are explorers. Blue dots are recruiters, and Red dots are soldiers. The enemy target is represented by the green dot.

In the point-attack and point-defense scenario, our goal was to explore the evolution of the agents' state switching behavior. In this simulation, however, our focus was less on how the agents switch as it was on allowing the swarms to evolve recruitment methods. In order to simplify the evolution process, the weighting functions that define the agents' behavior within the three states 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 is 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. These states are shown in Figure Seven.

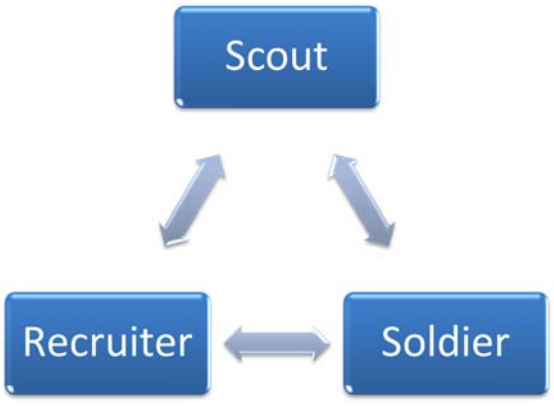

Figure Seven. The three possible states in the Search and Destroy swarm.

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 and 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 ends 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 vs. dweebs" scenario [3] 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 thesis are designed with the goal of providing useful techniques, the flexibility of the swarm is maintained by allowing the weighting functions to be evolved.

3.4.2 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 toward our goal, while at the same time allowing the swarm to find unexpected results. For the fitness function, if the swarm at least finds the target, it receives one point. A swarm can also receive up to 100 points depending on how much damage it inflicts on the target. The total distance traveled 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 traveled. Less movement is 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 within the playing area.

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 criterion used 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 receives 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.

3.4.3 Results

After evolution, the swarm successfully learns to 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 traveling 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 hope of improving the effectiveness of the search. Agents 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 and destroy simulation, while simpler than the point-attack and point-defense swarms, demonstrates 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 point system comparison between teams is shown to be a useful method to invert the swarms.

3.5 Base Attack

3.5.1 Simulation

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 appear at the edges of the playing area. These enemy units periodically fire projectiles toward 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. An 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-groups: scouts and recruiters.

Similar to the point-attack and point-defense swarms, the switching behavior of the agents is modeled as a threshold function. The threshold functions determine the probability 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 consider the base unguarded when, in fact, it was guarded, although the defenders were out of those agents' sensor 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 and 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 sufficiently large 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, groups of fewer than three agents will fail to destroy the enemy target.

3.5.2 Evolution

For the evolution of the swarms' parameters, we looked at a variety of evolutionary strategies [4][11][12][13][14][15]. 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 [8] or chess [7]. We were able to use this proven strategy to optimize our swarms' parameters. 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.

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. This process is described in pseudocode in Figure Eight.

```
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
        repeat
```

Figure Eight. Pseudocode for Evolution Process.

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 [9].

3.5.3 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 Nine. 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 more 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 Ten and 11 demonstrate the dynamic state switching behaviors learned by the swarms.

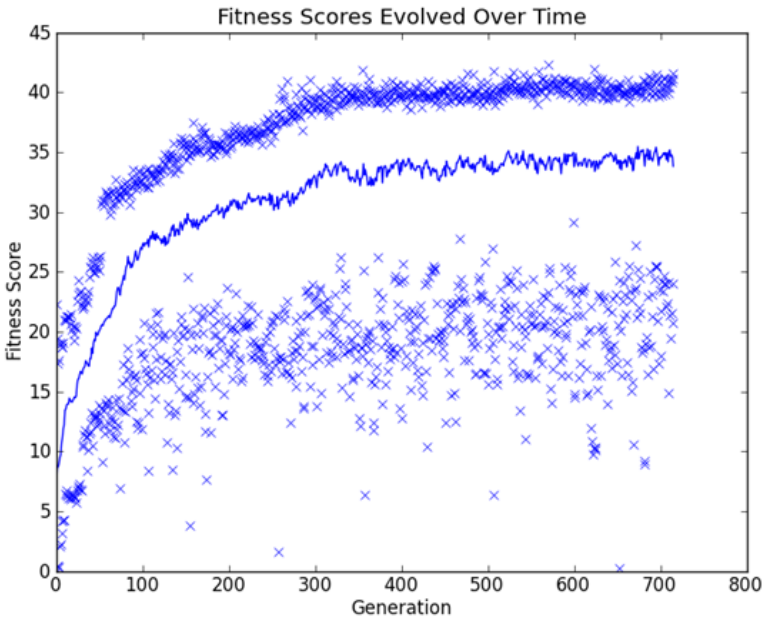

Figure Nine. This figure represents the learning process of the evolutionary algorithm by showing how the fitness scores improved over time. The solid line indicates the average fitness score of the population for each generation. The X's represent the maximum and minimum scores in the population. After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

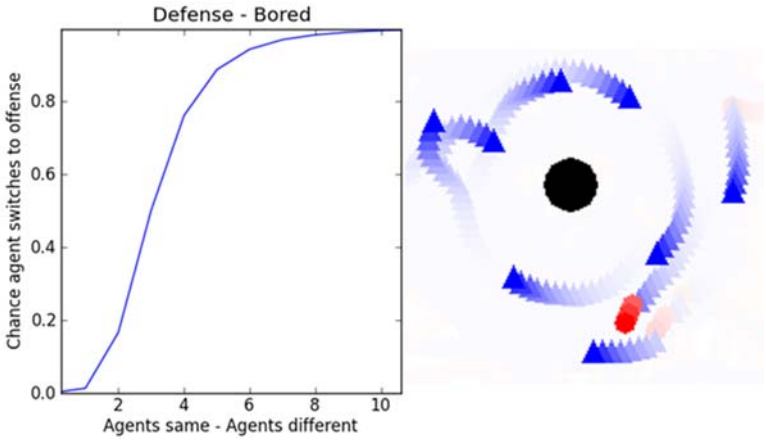

Figure Ten. The threshold function shown here demonstrates role saturation, which occurs when agents switch states after deciding there are too many units working on their task. First, the agent counts up the number of nearby agents working on its task and those working on a different task. This is fed 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 three, 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.

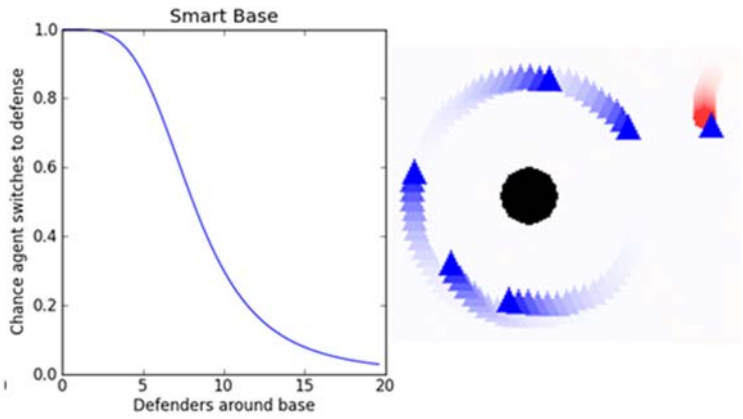

Figure 11. This function represents the way the agents process the information broadcast from the base. The base will broadcast the number of defenders around it and the agent has the option of switching to a defensive mode if it decides there are not enough defenders around it. In this case, the swarm will attempt to keep at least eight 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 six defenders around it. A red scout has heard the message and decided that six 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. 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 as well if they were too close. Figures 14 and 15 show the defensive strategies used by the swarms.

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.

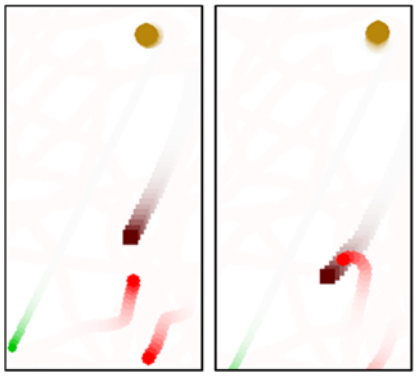

Figure 13. In this screenshot, a red scout approaches a maroon recruiter. While the scouts are inclined to stay away from other agents when they are relatively far apart, their attraction to recruiter overwhelms the initial repelling force. When the recruitment sensor brings the red scout close enough 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.

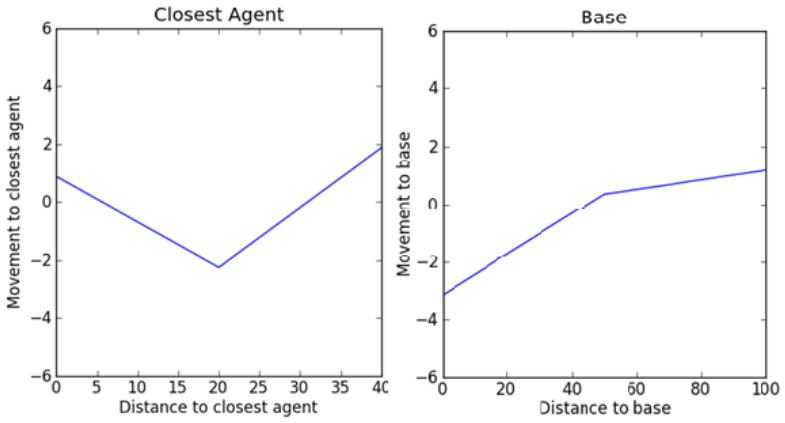

Figure 14. This weighting function shows how defensive agents will stay approximately 31 units away from each other. The zero crossing with a positive slope creates a "sweet spot" that the agent is inclined to stay in. Note, the function does become positive for small distances, but since the function is negative for values from six 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.

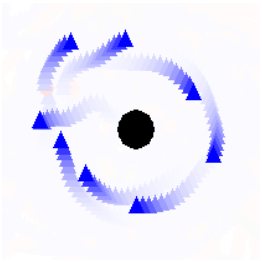

Figure 15. The defending agents learned to surround the base and travel in a circular pattern in order to intercept as many enemy projectiles as possible. They are also keeping a set distance away from each other so that if one detonates, it doesn't take out friendly units.

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.

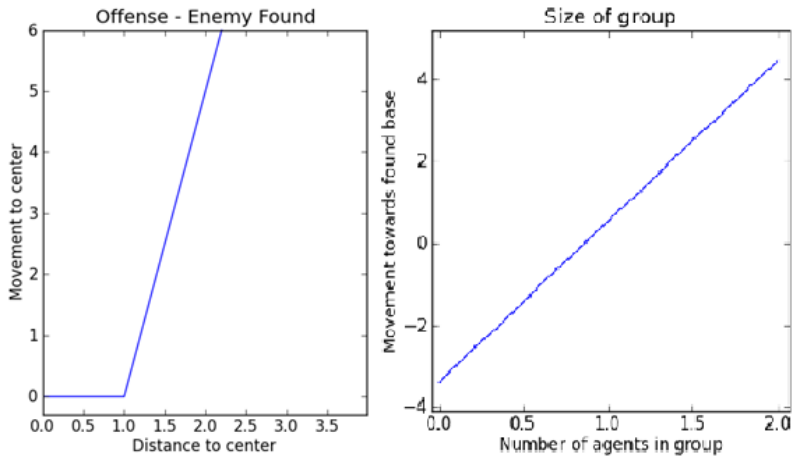

Figure 16. The first function shows the center sensor for recruiters. Note the scale for the x-axis. Any recruiters that are more than 1 unit away from the base will be inclined to return to the base. In other words, all recruiters return to the base. It is easy to understand why this unexpected strategy 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.

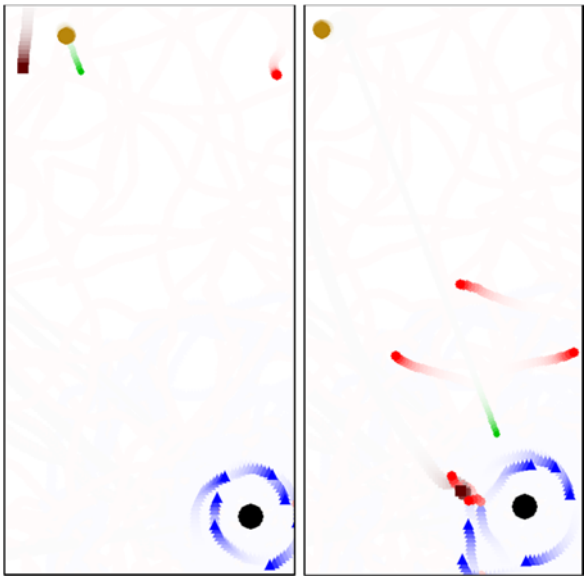

Figure 17. Here, a red scout has found an enemy unit and switched states to become a maroon recruiter. The recruiter is headed back toward the base in order to recruit other agents to form a group large enough to take out the enemy. This image shows the recruiter being joined by a third agent, which 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 Chapter four.

3.6 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 swarms are required to perform well in two objectives. For instance, some swarms have to both defend a friendly target, while also finding and destroying enemy units. By using an evolutionary learning algorithm, the weighting functions that defined the swarms' behavior are optimized to be successful in both the offensive and defensive objectives.

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 section 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.

CHAPTER FOUR

The Effects of Sensor Failure on the Emergent Behavior of Multi-state Swarms

4.1 Introduction

Swarms in nature are composed of a large number of individual agents, such as bees or ants, that each follows a set of basic rules [1]. Often these rules can be described as a simple cause-effect relationship between external stimuli and an agent's response. Each agent's response contributes to the emergent behavior of the swarm, which can often be unpredictable [2][3][6]. However, what if agents were unable to sense a certain stimulus? For example, what if a group of drones was unable to sense the difference between friend and foe? Or what if they had difficulty sensing certain types of enemy units altogether? How would that reduction in awareness affect the emergent behavior of the swarm?

The robustness of disjunctive control, also called Combs control [17][18][19], has the advantage of seamlessly recovering from failed sensors by exploiting possibly redundant information available from other sensors when available [20][21][22]. A simple example is steering a car to the right. This can be done by a) turning the front wheels of the car to the right, b) turning the back wheels of the car to the left, c) braking the two wheels on the right side of the car, or d) accelerating the rotation of the two wheels on the left side of the car. As long as one of these control actions exists, the car can be steered to the right even when the remaining three functions fail. Redundancy in the information available in sensors is often not as obvious as in this example. In the case

of swarms, the failing of one or more sensors may prompt the swarm to adopt a different emergent strategy in order to meet the objective of the collective.

Our goal is to discover what new emergent behaviors occur when certain capabilities are removed by disabling some of the agent's sensors. A previously tested simulation is used as a baseline for this experiment. In this chapter, we demonstrate the emergent behaviors that result from individually removing four of the agents' sensors.

4.2 Swarm Intelligence

In the simulation, a swarm is tasked with completing two objectives: guarding a central base from enemy attacks while also searching out and destroying enemy units [16]. In order to encourage recruitment, at least three agents were required to successfully kill an enemy unit. The simulation is ended when the base sustains a set amount of damage: in this case, when ten enemy projectiles hit the base. The movement of the agents is controlled by sensors connected to weighting functions. When an agent senses an object, the distance to that object is fed into a function that tells the agent how to move with respect to the object. In addition to a series of object sensors and their corresponding weighting functions, agents are also equipped with a center sensor that tracks how far away the agents are away from the base and pulls them back in if necessary. All of the sensors have limited range, except for this center sensor.

In order to accomplish both objectives, the agents are allowed to take on one of multiple states: defender, scout or recruiter. For each state, there is a different set of sensor weighting functions, including the center sensor functions. When an object switches states, it is changing the set of rules it follows. Since the swarm needs to dynamically adjust its resources, the agents are able to switch between states using

threshold functions. If an object senses there are too many agents working on the same task, or too few, the agents are able to change states or request other agents to change states as necessary. How the agents make this decision is determined by the threshold value. Each weighting function and threshold is an adjustable parameter that represents a rule that the agents follow. These rules can be optimized through the use of an evolutionary algorithm. This chapter discusses what happens when one of these rules is removed from the swarm's decision making.

4.3 Evolution Process

The swarm's performance is maximized using an evolutionary learning algorithm [4][11][12][14][15][17]. Performance is measured by tracking the number of enemy kills and the swarms' fitness scores. Each swarm is assigned a fitness score which is equal to the time survived, multiplied by the percentage of surviving agents within the field of play, multiplied by an efficiency factor that represents the reduction of the distance traveled by the swarm, as shown in Equation One. E is the efficiency factor and C is a constant.

$$F = Time * P_{agents} * E * C$$

Equation One

An initial, random population of teams, each represented by a set of weighting functions and thresholds, is evolved by simulating the scenario with each member of the population and comparing the results. Teams with higher kill totals and fitness scores survive and advance to the next generation of evolution, while the poorer teams are removed. Each surviving team is copied and mutated by adding random Gaussian noise to the weights and thresholds. For weights with values ranging from negative five to

five, the standard deviation of the Gaussian noise added is approximately 1.25. This algorithm allows the population of swarms to "learn" which rules and strategies successfully accomplish both objectives of attacking and defending [8][9]. This process is repeated for approximately 800 generations, resulting in a set of weights that are optimized for the given scenario. A brief description of the emergent behaviors is given below.

To accomplish the first objective of defense, defenders learn to loosely circle the base and intercept enemy projectiles by moving between them and the base. If the number of defenders drops too low, nearby scouts switch tasks to help defend the base. For the second objective, scouts learn to spread out from the base and from each other while looking for enemy units. When an agent finds an enemy, it switches to a recruiter and returns to base. At the base and along the way, scouts join up with the recruiters until the recruiter senses that the group is strong enough to destroy the enemy unit. The recruiter leads the group back to the enemy in order to eliminate the enemy. Any surviving agents in the group then go back to exploring the map for enemies.

Now that we have a solution for the multi-task scenario, our next step is to begin disabling sensors one at a time in order to determine how the failure of one of the sensors would impact the emergent behavior of the swarms. The first sensor that is removed is the projectile sensor. This sensor is used by defending agents to see incoming enemy projectiles that are headed towards the friendly base. Defenders are normally able to intercept the projectiles. We want to see what happens if the defenders are no longer able to sense these projectiles. The population of swarms that were evolved with all of their

sensors intact is used as the initial population for a second round of evolution without this projectile sensor.

Next, this process is repeated. The projectile sensor is turned back on and the group sensor is removed. The group sensor allows the agents to track how many other agents are nearby. This is used in recruitment since it allows the recruiting agents to know if they have enough friendly agents around them to destroy an enemy unit. The evolving population is reset to the initial group of teams that were evolved with all their sensors and the evolutionary algorithm is run again. This is repeated two more times: once with the center sensor broken and again without the base sensor.

4.4 Results

Fitness scores for each of these four evolutionary cycles are shown in Figure 18. In each case, the fitness scores are lower with the sensor removed. However, these scores do improve over time as the swarms' weights are adjusted to maximize the use of the remaining sensors the swarms does have. Even though it is missing one of its sensors, in some cases a swarm is still able to achieve scores almost as high as a swarm with all sensors available. One of the main differences in emergent behaviors that develop is how the defending agents guarded the base. These strategies are depicted in Figure 19.

4.4.1 Projectile Sensor

The results of removing the projectile sensor area shown in Figure 20. Removing the projectile sensor does not greatly affect the fitness scores, which drop slightly compared to the results of the previous evolution. This is because the swarms are able to adapt to the loss of the sensor and still accomplish their objectives similarly to before.

Figure 18. The results of the evolution before and after breaking the four sensors are shown here. Before generation 200, the swarm is evolving with all of its sensors. At the discontinuity, the graph splits to represent the fitness scores after the removal of the sensors. The swarm performs well with its projectile and base sensors removed and okay without its group sensor, but very poorly when the center sensor is disabled.

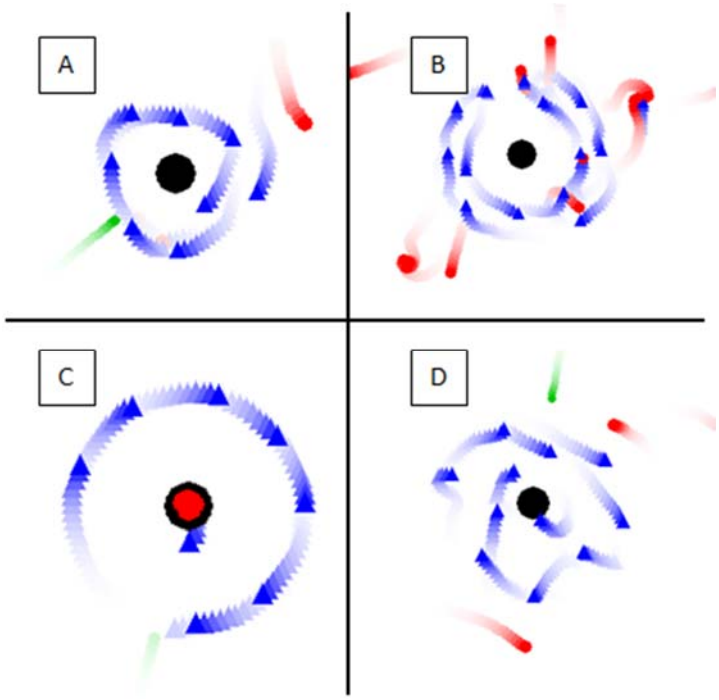

Figure 19. This figure demonstrates the emergent behavior of defending agents evolved with different sensors disabled. (A) shows the tight ring formed when the projectile sensor is removed. (B) shows attackers forming groups before scouting due to the fact that the group sensor is off. (C) shows the ring formed around a group of replacement scouting agents when the center sensor is disabled. (D) shows a mob of defenders surrounding the base with the base

With the sensor, defending agents circling the base are attracted to projectiles and move in for the kill. Without the sensor, defenders are unable to see the projectiles. The swarm's solution is to make its circling behavior tighter and faster. By rapidly circling around the base, agents are able to intercept most enemy projectiles simply by running into them. The tighter circle uses a smaller group of defenders to effectively defend the base and allows more scouts to look for enemy units. This is an example of a behavior that existed previously, but was adjusted in order to compensate for the loss of a sensor. The behavior of the scouts and recruiters remains the same, since their task does not involve defending the base from projectiles.

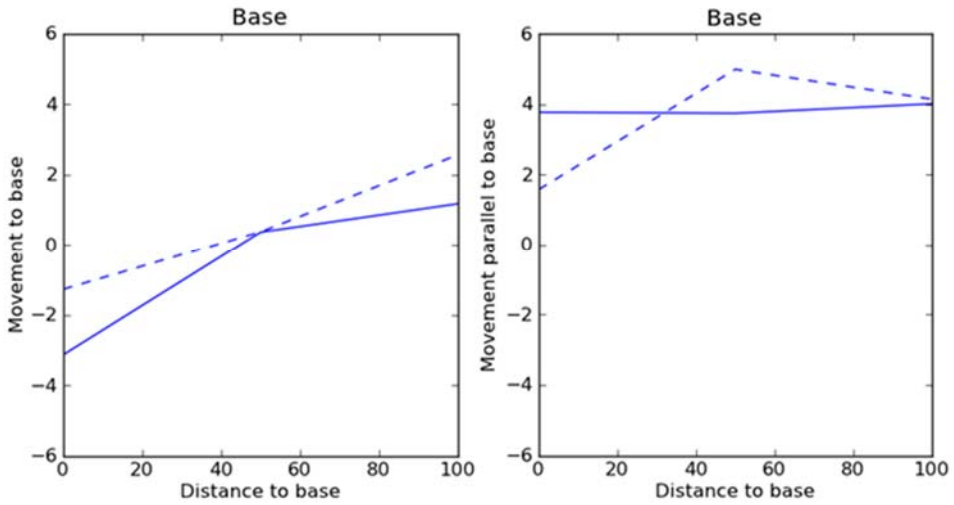

Figure 20. Without the projectile sensor, the swarms adapt by tightening the ring around the base and speeding up their rotations. In the first graph, the old strategy is represented by the solid line and the new function is the dotted line. Initially, the "sweet spot" that the defenders converge to is around 50. This is lowered to a radius of 40 by the new method. In graph 2, the speed at the "sweet spot" (40) ranges from a little under four to almost five.

4.4.2 Group Sensor

The group sensor is proven to have more of an effect on the swarms' behavior. This sensor is crucial for the recruitment techniques developed previously since it allows

recruiters to count the number of nearby agents. When this ability is taken away, the agents are no longer able to determine if their group is large enough to destroy an enemy. Through the evolutionary process, the swarms respond by modifying their strategy, as shown in Figure 21. Instead of scouts spreading out, over time the scouts form up in clumps and scout in groups. This removes the need for recruiters to return to the base to find other agents. When an enemy is found by a group, recruitment is no longer needed. Instead, assuming it is a group of at least three agents; the group can simply move in immediately and destroy the enemy. The defensive strategy remains the same, but with a smaller number of defenders to allow more scouts to explore. Compared to the original evolution, scores drop since the effectiveness of the search is reduced when the scouts search in groups. However, considering the previous recruitment strategy no longer works at all, this emergent behavior is a reasonable alternative, allowing the swarm to still find and destroy enemy units while maintaining the previous defensive strategy of surrounding the base in a rotating ring.

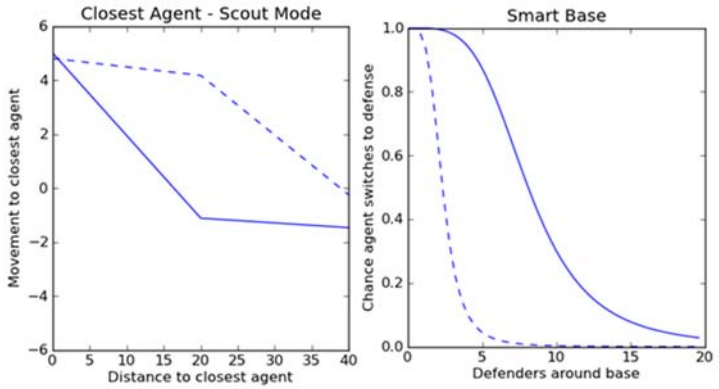

Figure 21. Since the group sensor is disabled, the swarms have to adapt. Originally, scouts are repelled from each other. The value of the solid line is positive (indicating attraction) at small distances. However, since the agents are repelled at larger distances, the agents should never actually get close enough to each other for that to matter. The new function causes agents to be much more attracted to each other. At large distances the attraction is weak, but still positive, and increases as the agents move closer. The second graph shows how many defenders the base is expecting. The swarm learns to use a smaller number of defenders in order to allow more agents to explore.

4.4.3 Center Sensor

Unlike previous attempts, when the center sensor is removed, the swarm scores drop off drastically. It turns out, in this particular scenario, the center sensor is necessary for the success of the simulation. Without it, there is nothing to prevent exploring agents from wandering off the edge of the map. Since the swarm is unable to perform well in its task of finding and destroying enemies, it focuses on the second task of defending its base, as shown in Figure 22. The unexpected emergent behavior that develops from evolving without this sensor is the clumping of scouts in the base. When the simulation begins, some of the scouts form groups and begin searching, but most of the scouts form one large group and converge right on top of the base. These scouts, while not scouting, are used as reserves to replace destroyed defenders. The defending ring is similar to before, but with a much larger radius and a group of eight evenly spaced defenders surrounding the base. As the defenders detonate to take out incoming projectiles, they are replaced by scouts in the base. The defenders are far enough away from other defenders and the base (which is full of scouts) so that detonations do not take out any friendly units. This strategy allows a large portion of the swarm to stay defending the base which keeps the base alive for as long as possible. While the swarm is unable to properly complete both objectives, this modification does allow the swarm to perform well in at least one.

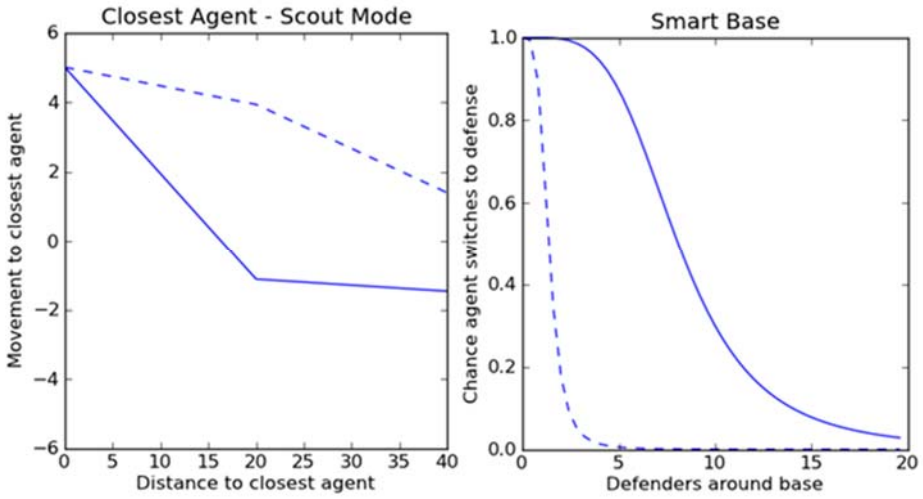

Figure 22. These graphs look similar to those in Figure 21. However, there are a couple main differences. On the left, the new strategy developed without the center sensor is represented by the dotted line. Scouting agents are always attracted to each other since the function is positive for all distances, including an attraction of 1.5 at the maximum distance of 40. This is a much stronger attraction than that in Figure 21. Also, the second graph shows how the swarms have learned to use fewer defenders, even fewer than when the group sensor is removed.

4.4.4 Base Sensor

Without the base sensor, the swarms are able to still achieve fitness scores on par with previous results. Since the defenders are unable to sense the base, the previous rotating ring strategy is impossible. To compensate for this loss, the defenders form a mob and surround the base, as shown in Figure 23. The defenders also learn to spread out from each other instead of linking up in the circling method. Previously, defenders used the attraction from the base sensor to stay close to the base. After evolution, the swarm learns to use the center sensor to accomplish the same objective by pulling in defenders that wander off more than 50 units away. This group of randomly moving defenders makes it difficult for enemy projectiles to reach the base, even though it is not as efficient as the ring strategy. The scouting and recruiting methods are not affected by the loss of the base sensor, so overall fitness scores remain high.

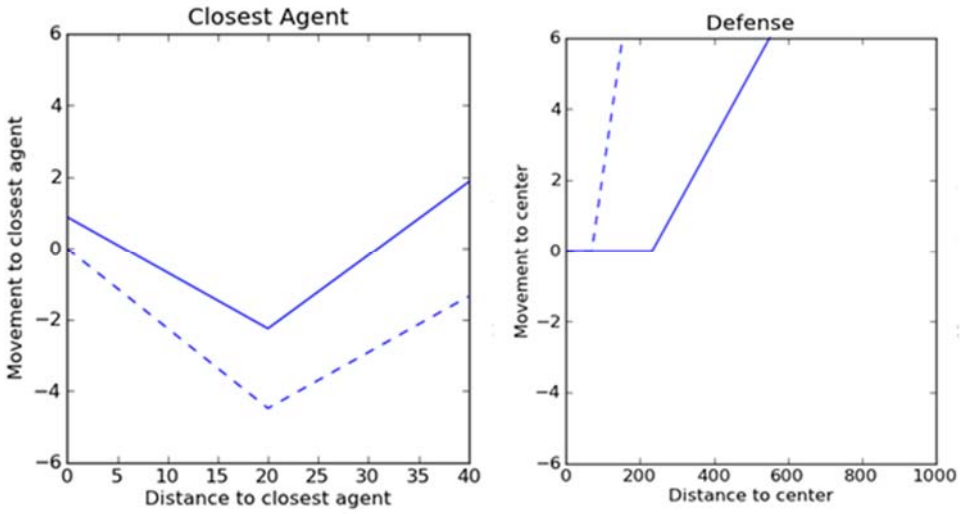

Figure 22. These graphs represent some of the changes in rules that develop when the base sensor is turned off. The first weighting function shows how defenders react with respect to other agents. Initially, defenders are attracted to the "sweet spot" at the zero-crossing of the graph: around 30 units away. After evolving without the base sensor, the defenders learn to spread out and are always repelled based on the negative value of the function. The second graph shows the changes in the center sensor. Initially the sensor pulls back defenders that get more the 250 units away from the base. After evolution, the defenders are kept in a tighter radius of closer to 50.

4.5 Conclusion

We have shown that while decreasing a swarm's abilities will negatively affect its performance, an evolutionary algorithm can allow the swarm to learn a new strategy that utilizes its strengths. In four separate cases, a swarm is evolved to complete a multi-objective scenario, each time with the loss of one sensor. Sometimes an emergent behavior is refined to maintain swarm performance. An example of this is when the rotating defenders formed a smaller, tighter circle around the base to better defend it when their projectile sensors were removed. Other times, a completely different strategy needs to emerge, such as when the scouts began exploring in groups to remove any need for recruitment. Also, if a swarm is unable to accomplish one objective due to sensor loss, then it puts more of its focus into the other objective. For instance, disabling the

center sensor greatly reduced the swarm's ability to search for enemies, so instead the swarm began to keep a large percentage of its agents near the base in order to survive for as long as possible. In each case, even though the fitness scores drop initially, the swarms are able to use other sensors in order to at least partially compensate for the sensor reduction.

CHAPTER FIVE

Mapping an Underwater Minefield with a Multi-State Swarm and the Effects of Size on Its Performance

5.1 Introduction

Another application of swarms in warfare is minefield mapping. The problem of clearing a minefield has been studied throughout the past hundred years and recently the use of swarm intelligence has been shown to be a viable option [23][24]. However, for the most part, much of the work has dealt with underground mines. In these cases, the minefield can be marked by agents in a swarm to communicate to other agents that an area of the field has previously been searched for mines thoroughly. This is similar to how ants forage for food. When an ant finds a source of food, it returns to the anthill, releasing a trail of pheromones as it goes. Other ants are able to follow this pheromone trail to the food. While this emergent swarm behavior is interesting and can be applied to underground mine scenarios, it does not function well when dealing with underwater mines. In an underwater scenario, AUVs are unable to leave a trail that would be unchanged by local currents, marine life, etc. Since it would be unfeasible to mark trails underwater, a new strategy is needed.

5.2 Minefield

In this scenario, a single swarm is tasked with searching an area for mines and reporting all the mine positions to the central base, as shown in Figure 24. Again, since we are working with multi-state swarms, agents are able to take on one of three states: explore, report and recharge. All agents are initialized as explorers. When an agent

approaches an unreported mine, the agent has the option of switching to a reporting state. When a reporter reaches the base with the location of a mine, that location is saved and that mine marked so that other agents that see that mine know that it has already been mapped.

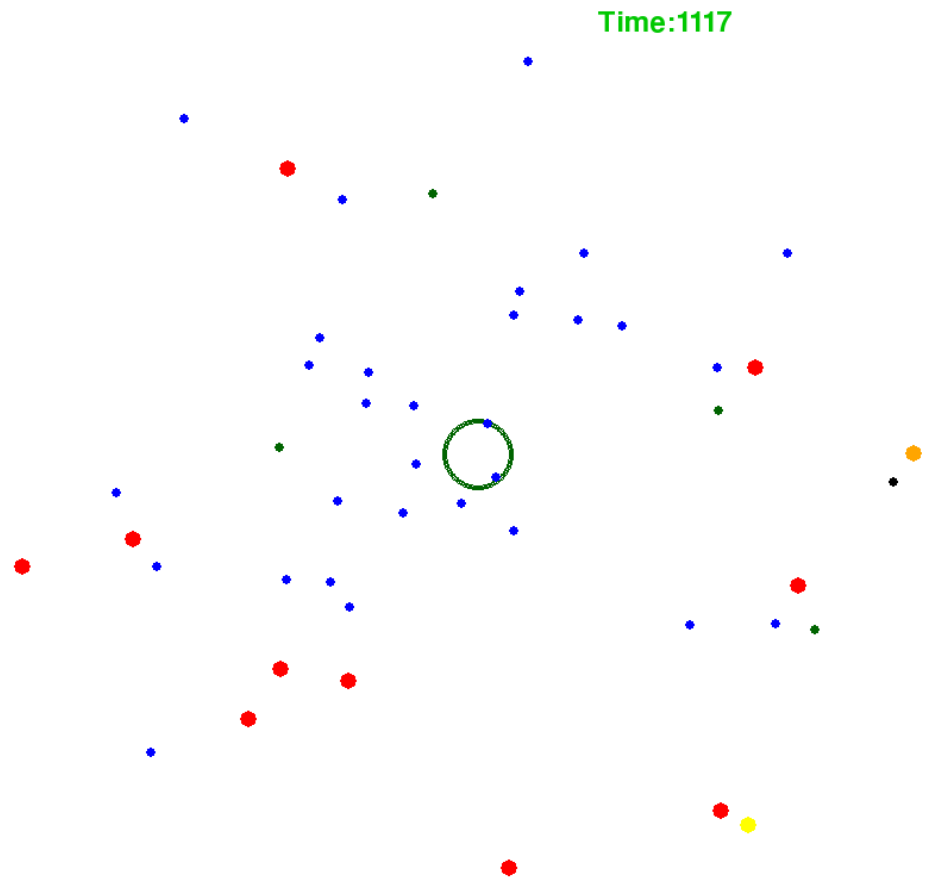

Figure 24. Screenshot from Minefield simulation. Blue dots represent agents in the explore mode. Green dots are in the recharging state and the black dot is acting as a reporter. The large, red dots show the locations of all of the reported mines. The orange dot is a mine that has been found (by the nearby reporter) but has not been reported yet. The yellow dot is a yet undiscovered mine. The ring in the center of the map represents the sensor range of the central base (not pictured) as well as the recharging radius for the agents.

We also introduce a recharging state for the swarm. This concept of battery life is not considered in our previous work [2][3][16], but was something we wanted to add in

order to approach a more real-world applicable solution. All units are initialized with a battery charge of 100. After each cycle, the battery is decreased in two ways. First, there is an idle battery discharge that occurs regardless of movement. Second, the battery is discharged based on how far the agent has travelled that time step. If an agent's battery drops to zero, then the agent remains stationary and is considered lost. Agents have the ability to switch from either exploring or reporting states to a recharging state, if their battery drops too low, as defined by the output of an evolved threshold function. Agents that are near the base are able to recharge their batteries. In practice, this could mean surfacing for solar cell recharging, refueling from a ship, or other possibilities, but for simplicity, agents in this scenario are able to recharge if they are within a set radius of the central base. When the battery charge reaches an acceptable level, the agents switch back to the task of exploring the search space.

In evolving the swarm, a fitness function is needed to determine how well the swarm maps out the minefield. Our resulting fitness function is computed as a product of scores that represents how well the swarm achieves some specific objectives. The first, and most obvious, objective to track is how many mines the swarm is able to find. Our goal is for the swarms to find at least 90% of the mines. These goals are adjustable and simply reflect what we consider success. In practice, a swarm designer needs to define what a "successful" swarm means to them. Expecting 100% completion of a task 100% of the time may be setting the bar too high, so for our purposes, 90% is used. The second objective tracked is how many mines were reported. We do not want the swarm to simply find the mines. They need to report the mine positions to their home base as well, and 90% is set as an acceptable percentage for the purposes of evolution. Thirdly, the

swarm should keep as many agents alive as possible, so the percentage of agents remaining alive, within the field of play when time expired is also tracked. Agents are considered dead if either their battery level drops to zero or they bump into a mine. 95% is used as the success threshold here. Finally, we would like the swarm to perform its task as fast as possible, so the time remaining was used as well. The fitness score is computed using the following formula:

$$F = (1 + P_{Found}) * \left(1 + P_{Reported}\right) * (1 + P_{Swarm}) * (1 + \frac{T}{2000})$$

Equation Two

In the above equation, P_{found} is the percent of mines found, $P_{reported}$ is the percent of mines reported, P_{swarm} is the percent of swarm remaining and T is the time remaining (out of the total time limit of 2000 time steps, assuming all of the mines are reported). However, if any of the three percentages is greater than the set success threshold, the threshold value is used instead. In a successful evolution, the swarm's percentages of mines reported, mines found and swarm remaining should eventually increase to be greater than the predefined thresholds. When this happens, the swarm has a successful strategy, and the evolutionary algorithm begins optimizing the swarm's strategy for speed.

5.3 Results

Figures 25-28 represent the progress of improving these objective percentages over the course of the evolution process. Figure 29 shows the resulting fitness scores over the same period of evolution.

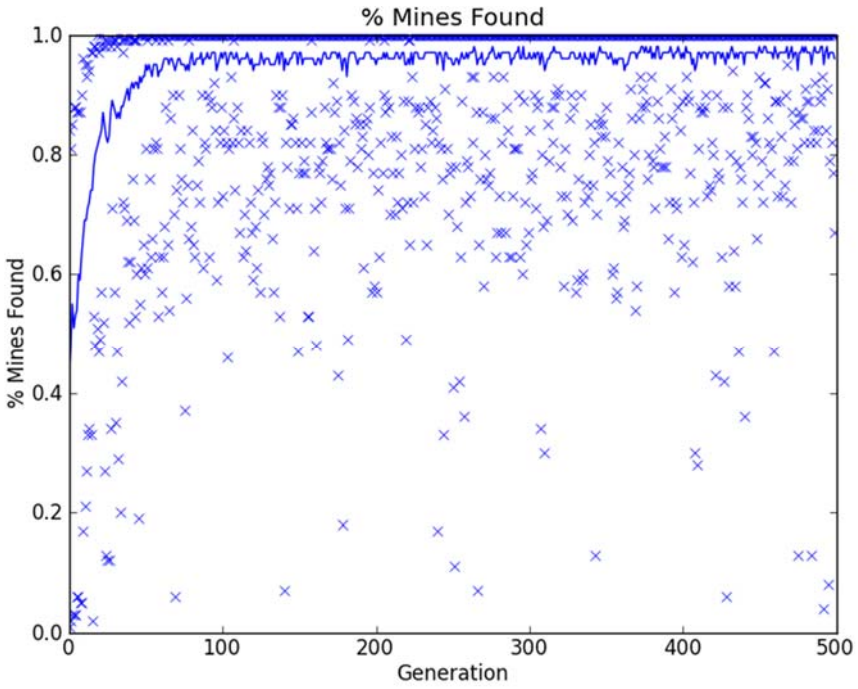

Figure 26. Mines Found. The average percentage of mines found reaches 97%, exceeding the goal of 95%. The maximum percentage found is 100%.

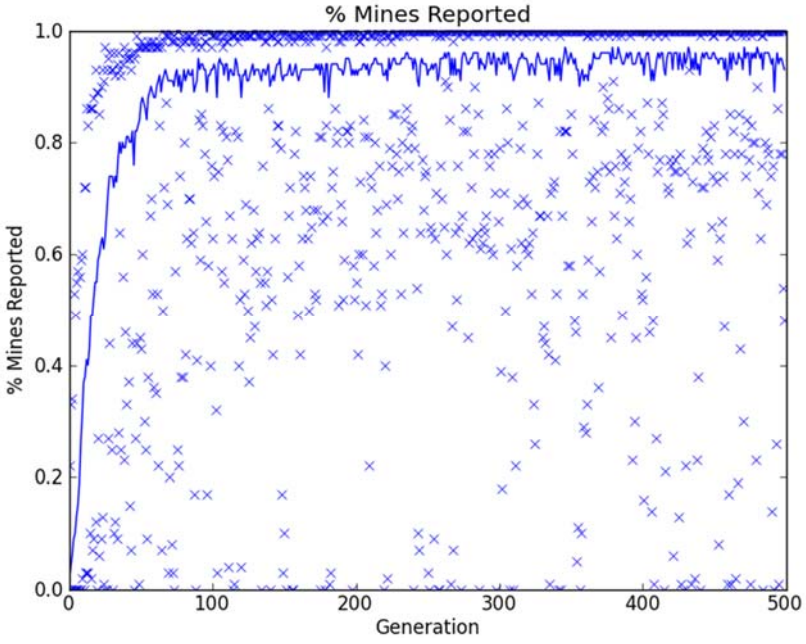

Figure 25. Mines Reported. The average percentage of mines reported for each generation is shown by the solid line. Maximum and minimum percentages for each generation are also indicated by x's. The average percent reported reaches the 95% threshold by the end of the evolution. The maximum scores approach 100% as well.

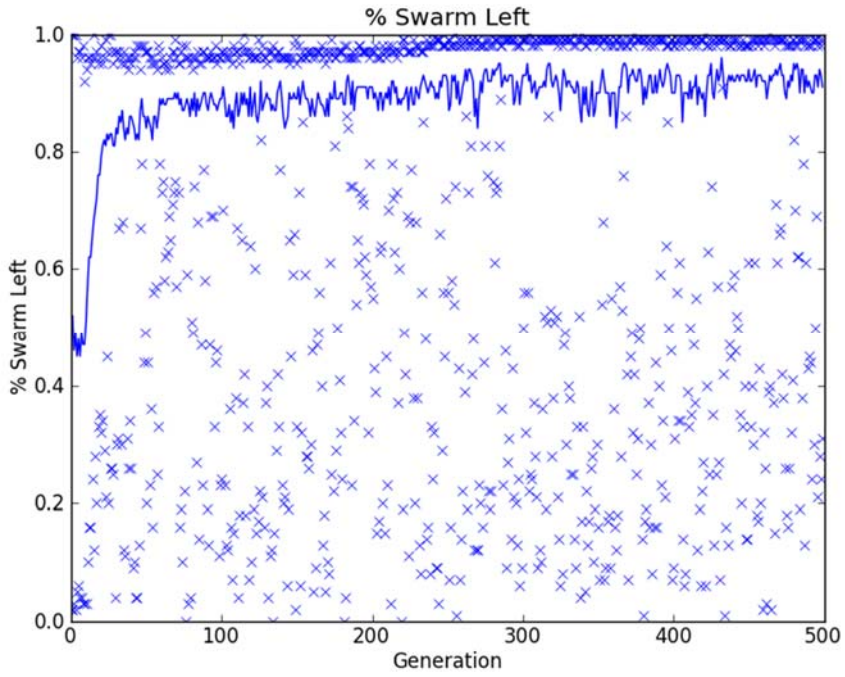

Figure 27. Swarm Left. On average, over 90% of the swarm survives the simulation by the end of the evolution process, which meets the requirement of 90% survival. Also the maximum percentage of the swarm remaining reaches approximately 99%.

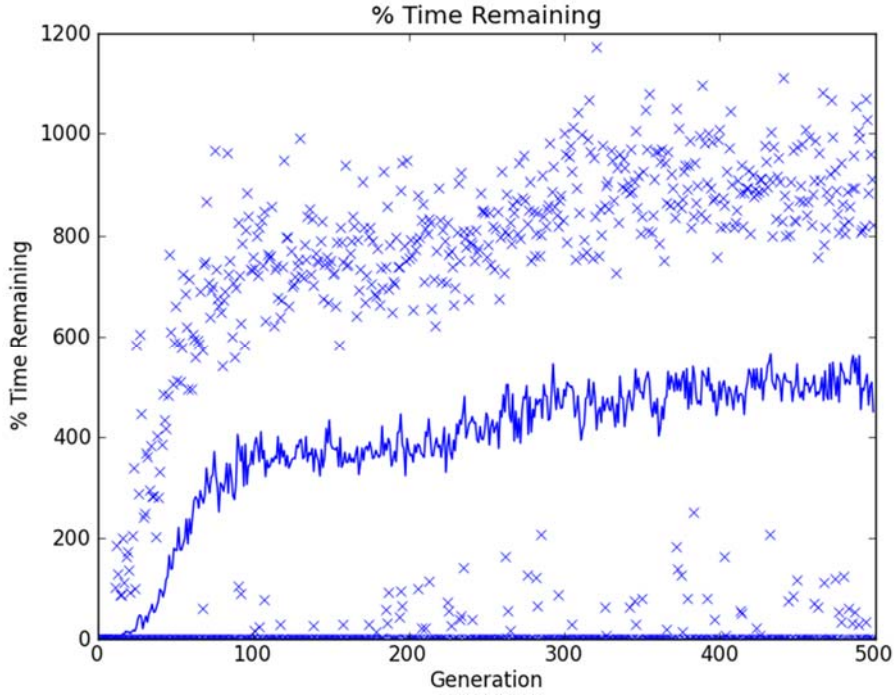

Figure 28. Time Remaining. The average time remaining at the end of a simulation converges to approximately 500. The maximum time remaining reaches 1000.

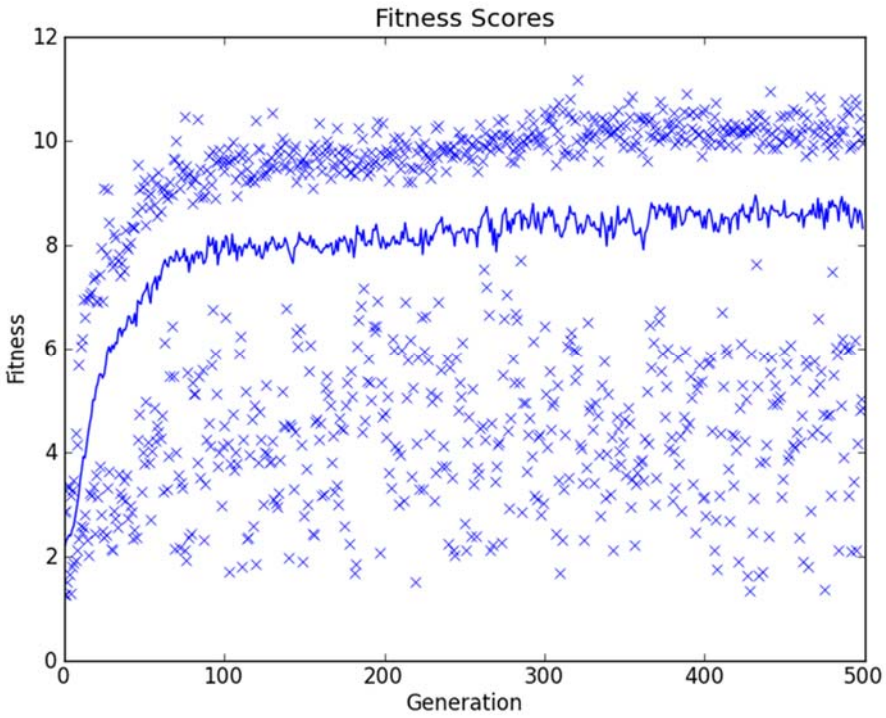

Figure 29. Fitness Scores. The average Fitness scores settle at 8.5, with maximum scores greater than ten.

When this simulation was first designed, the goal was to produce an emergent behavior similar to the leaf-cutter ant [1]. Certain species of leaf-cutter ants have a very efficient assembly line strategy to transport pieces of leaves. Instead of taking its piece of a leaf all the way to the anthill, each ant only carries its piece until it bumps into an ant headed the other way. The ant hands off the piece to the newcomer and returns to the leaf, while the newcomer turns back to the anthill with the leaf. This method is actually the most efficient strategy. Without these handoffs, the entire line of ants would be slowed down by the slowest ant.

Similarly, for our minefield simulation, agents were given different maximum speeds and reporters were given the ability to "hand off" information to nearby explorers, switching states with them. The decision making algorithm was controlled via a simple

three-input aggregator equivalent to a neural-network perceptron [26]. The inputs included the agent's maximum speed, its distance to the base, and its battery charge. Each input was multiplied by an adjustable weight and summed together. If the result was greater than a pre-set threshold, than the agent decided to perform the state-switching action. Reporters were able to request a handoff with an explorer and the explorers were able to decide whether or not to accept the request. If both the reporter and the explorer agreed to handoff the mine location information, then the agents would switch states with each other.

Unfortunately, it was discovered that this method of handing off bits of information does not improve the swarm's speed or efficiency in our scenario. In a two-dimensional grid, fast agents do not get stuck waiting behind slower agents. Instead of needing to hand of off their information to the slow unit, fast agents can just maneuver around the slower ones. This was learned by comparing swarms' fitness scores after being evolved both with and without these hand-offs. As shown in Figure 30, the results prove that there is essentially no difference between swarms evolved with and without this ability. This does demonstrate an inherent pruning ability of swarm inversion. Through analysis, hand-offs were shown to be a useless ability, and therefore may not be a necessary technological addition. The hand-off capabilities can be removed from the simulation to simplify the rules without negatively affecting the emergent behavior of the swarms.

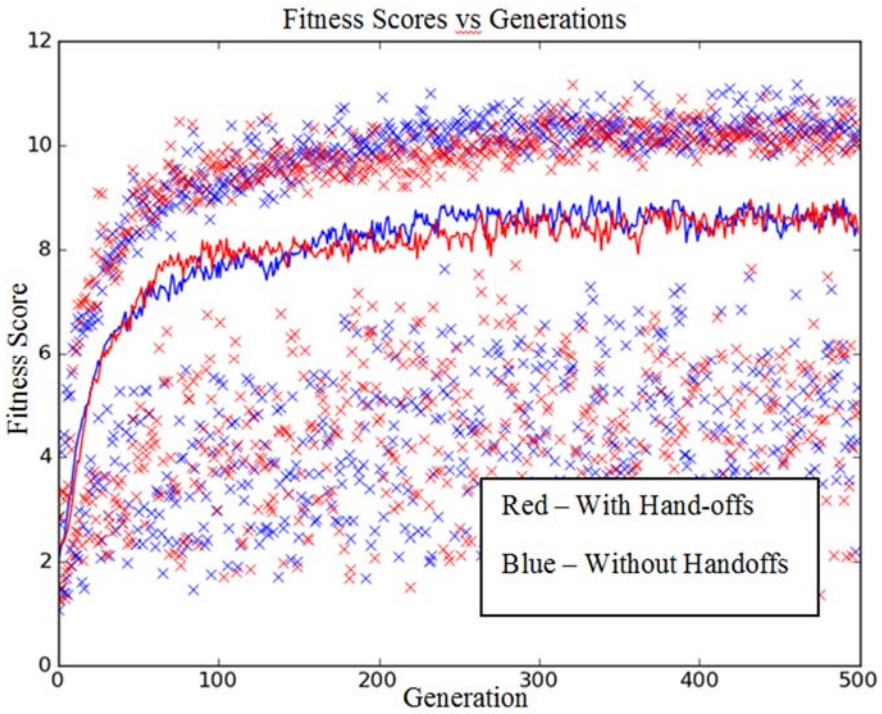

Figure 30. Comparison of Fitness Scores during evolution with and without handoffs. The red lines represent the scores generated by an evolution with handoffs. The blue show the fitness scores generated without handoffs. The results are essentially the same.

Even though the solution did not achieve the method we were expecting, the swarms are eventually able to find an efficient method of searching the area for mines. The emergent behavior of this minefield swarm is fairly simple, but produces a solution that successfully maps out all of the mine locations within the given time limit. Explorers spread out until they find a mine. If its location hasn't already been reported, the explorer switches to a reporting state and returns to the base in order to report the location of the mine. This is accomplished by reducing the center sensor radius to a small value for reporters. Agents also switch to their recharging state when their battery level drops too low. Recharging agents return to the base to recharge and switch back to explorers when the recharging is complete. Examples of these behaviors are shown in Figure 31.

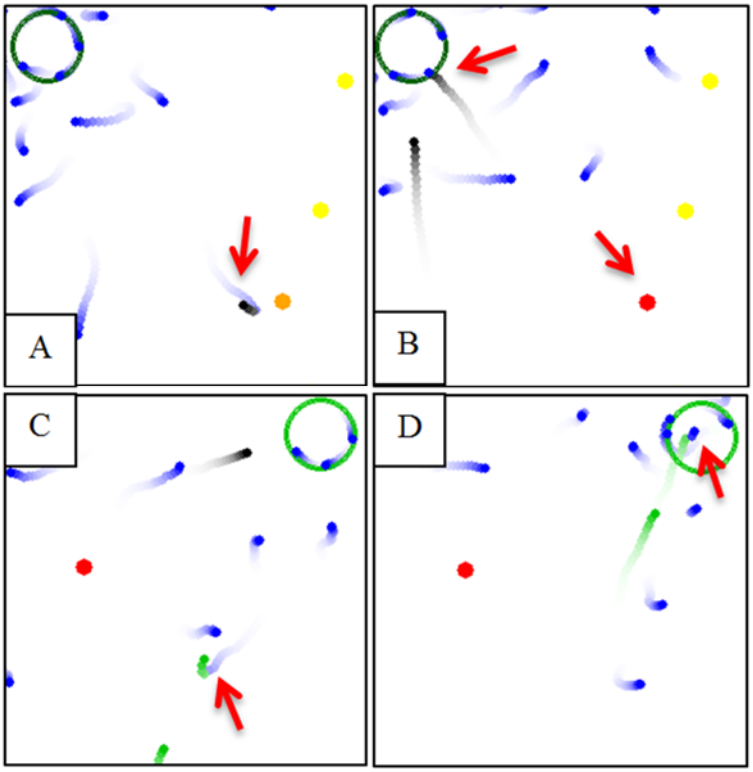

Figure 31. Examples of emergent behaviors. In A, a blue scout has just found an unreported mine. The mine has turned from yellow (unreported) to orange (found, but still unreported). The scout switches to become a black reporter and returns to base. In B, the reporter reaches the base and reverts to the blue scout mode. The mine also turns red, indicating that its position has been mapped. In C, a blue scout realizes its battery is too low and switches to a green recharging state. In D, it has finished recharging at the base, and becomes a blue scout again.

5.4 Swarm Size

One of the more "fuzzy" concepts in swarm intelligence relates to how large (or small) a group of agents should be in order to be considered a swarm. Obviously a group of two simple agents is probably not going to be able to perform any complex behaviors. Conversely, a group of a million agents would almost certainly be able to achieve its objectives, but the cost of having such a large swarm would vastly overwhelm the benefits. Finding the optimum number of agents to successfully and efficiently accomplish an objective is a useful problem to solve. We decided to test our Minefield

simulation with a wide variety of swarm population sizes in order to determine the best swarm size for our particular model and difficulty.

While designing the minefield scenario, it was initially tested with a population size of 30 agents. Our goal is to design a swarm that would be large enough to have complex emergent behavior, but still simple enough to be relatively inexpensive. In order to find the optimum swarm size, we evolve the swarm eight separate times with a variety of swarm sizes, ranging from fifteen to fifty. The scenario parameters such as map size, sensor ranges, time limit and number of mines, are kept the same for each simulation. The fitness scores across each of these evolutions are shown in Figure 32.

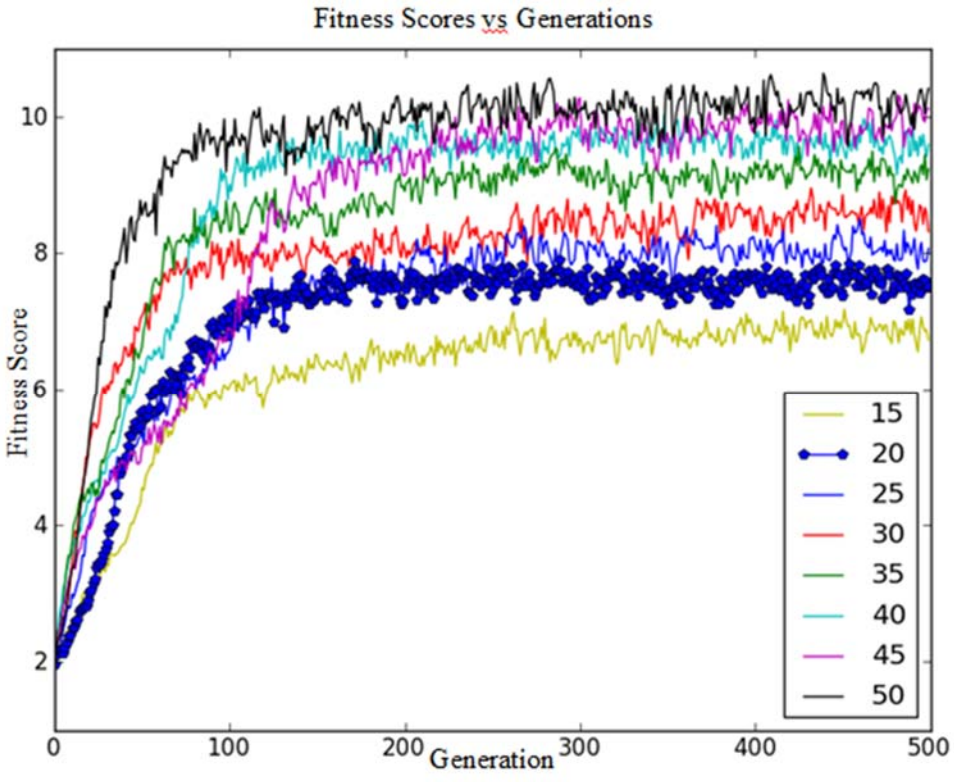

Figure 32. Comparison of fitness scores during several evolutions with different swarm population sizes.

As expected, the fitness scores increase as the population sizes increases. This is simply due to the fact that a larger swarm can more easily cover a larger search space. However, at some point we anticipate seeing diminished returns as we increase the swarm sizes. To test this, we take a highly evolved swarm solution from one of the evolutions (it did not matter which one, as all of the evolutions converged to the same solution) and test its performance when we changed the number of agents in the swarm. During the size test, the solution is tested one hundred times at each of the population sizes, which range from a swarm of one to sixty.

Figure 33 shows the increase of the fitness scores as the swarm size changes. As expected, we start to see diminishing returns and the swarm size becomes unnecessarily large. Figures 34 and 35 show the percentages of mines found and reported. It can be seen that a swarm of twelve agents can successfully meet the criteria of finding and reporting 90% of the mines. However, a 100 % success rate is not approached until the swarm size reaches 35 agents. After 35, there is not much difference in these two graphs. Figure 36 shows how the percentage of agents remaining changes due to differences in population size. While a very small swarm is able to keep the required 95 % of the swarm alive, it is not able to properly search the area for mines. It is not until the swarm size increases to around 12 that the swarm completes this objective and the mine search objectives. Another benefit of increasing the swarm size is shown in Figure 37, the speeding up of the swarm's success time. After a swarm completes the three required objectives, the next step is to try and perform those tasks as fast as possible. The time remaining increases rapidly around a size of 35 agents and slows down after that. Figure 38 provides a clearer insight into how swarm size impacts speed. In this graph, the time

remaining is divided by the swarm size, resulting in a peak that represents when the swarm is performing its objectives quickly and efficiently.

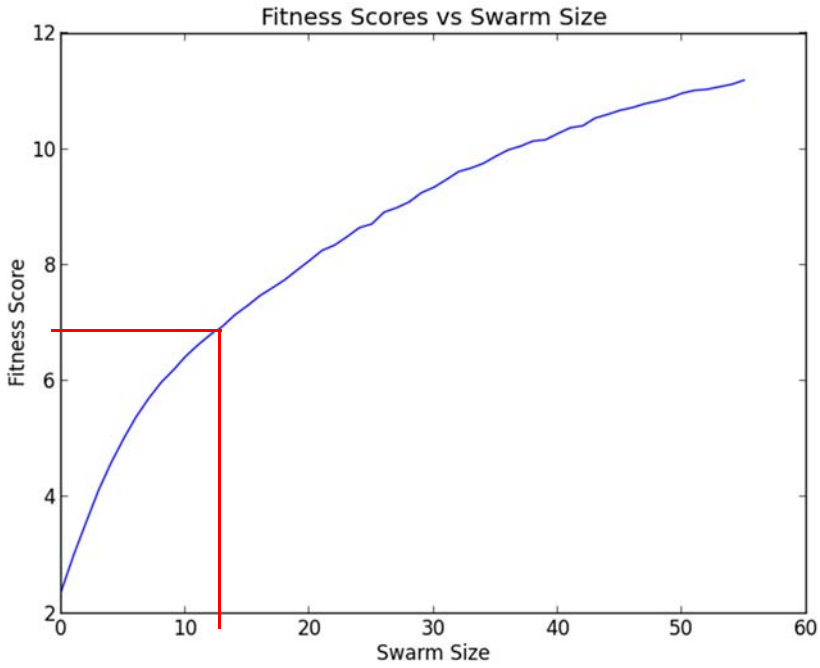

Figure 33. Fitness Scores vs Swarm Size. A score of at least seven represents a swarm that has successfully completed its objectives. Here, the fitness scores reach this threshold at a size of 13.

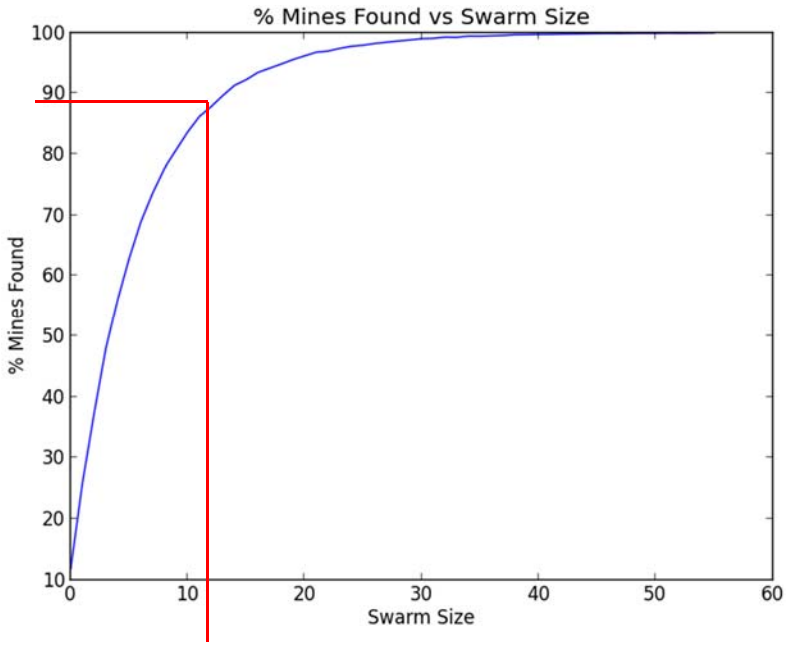

Figure 34. Mines Found. The percentage of mines found reaches the threshold of 90% at a swarm size of 13 agents, and begins experiencing noticeable diminishing returns around 35.

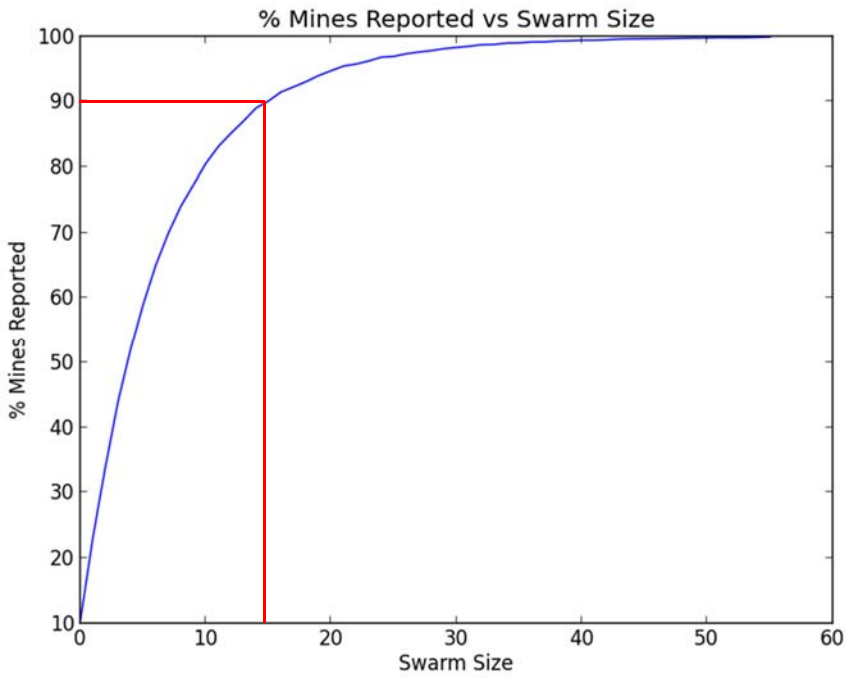

Figure 35. Mines Reported. Similar to Figure 34, a swarm of 15 agents is required to meet the threshold of 90% of the mines reported and again, diminished returns can be seen near 35 agents.

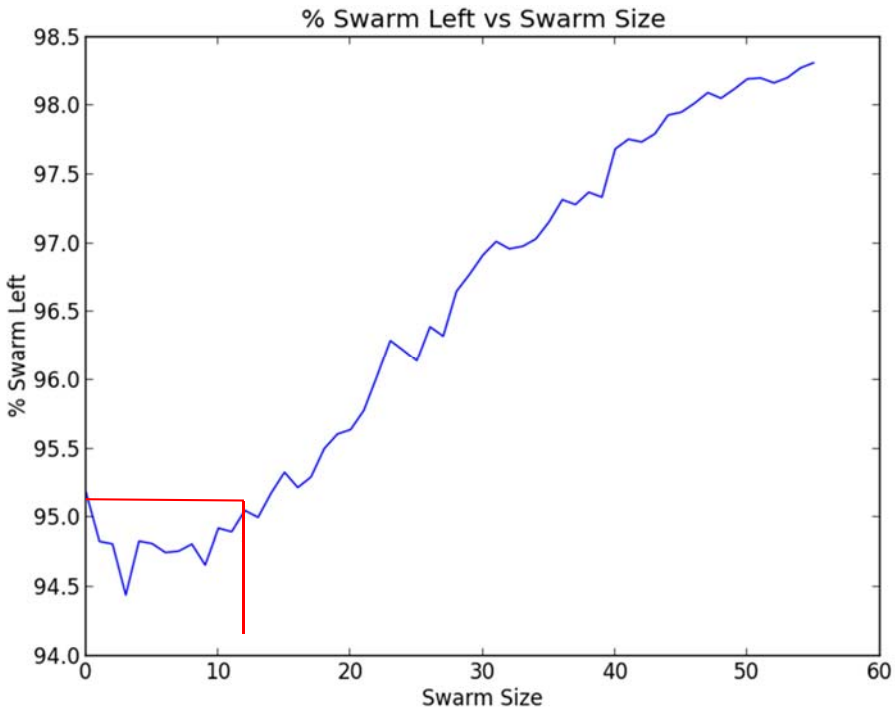

Figure 36. Swarm Left. 12 agents are required to meet the 95% survival criterion.

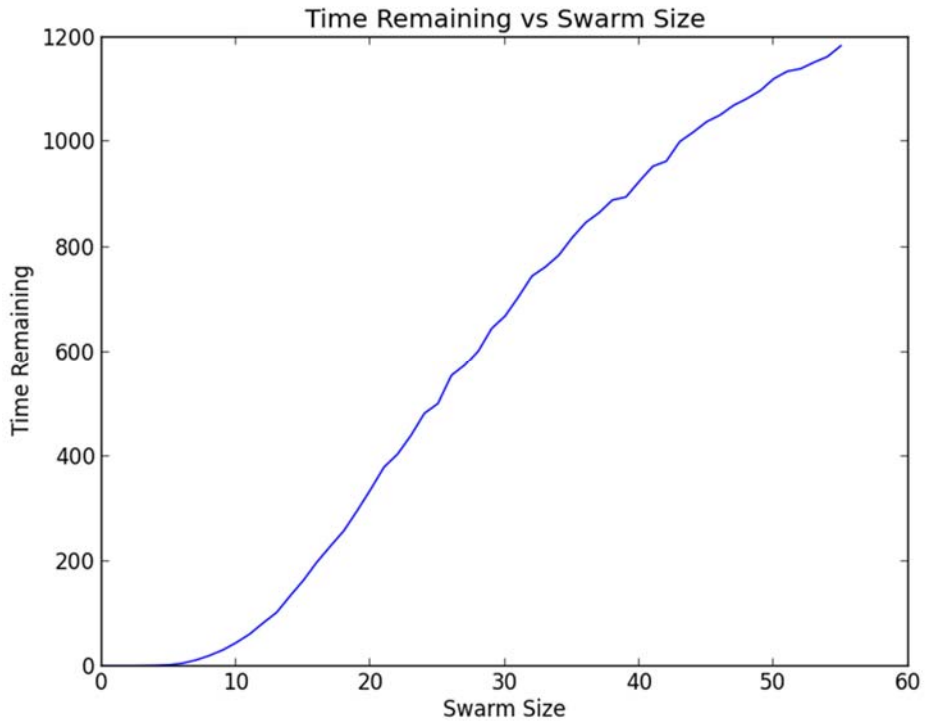

Figure 37. Time Remaining. The time remaining at the end of a successful search increases with swarm size.

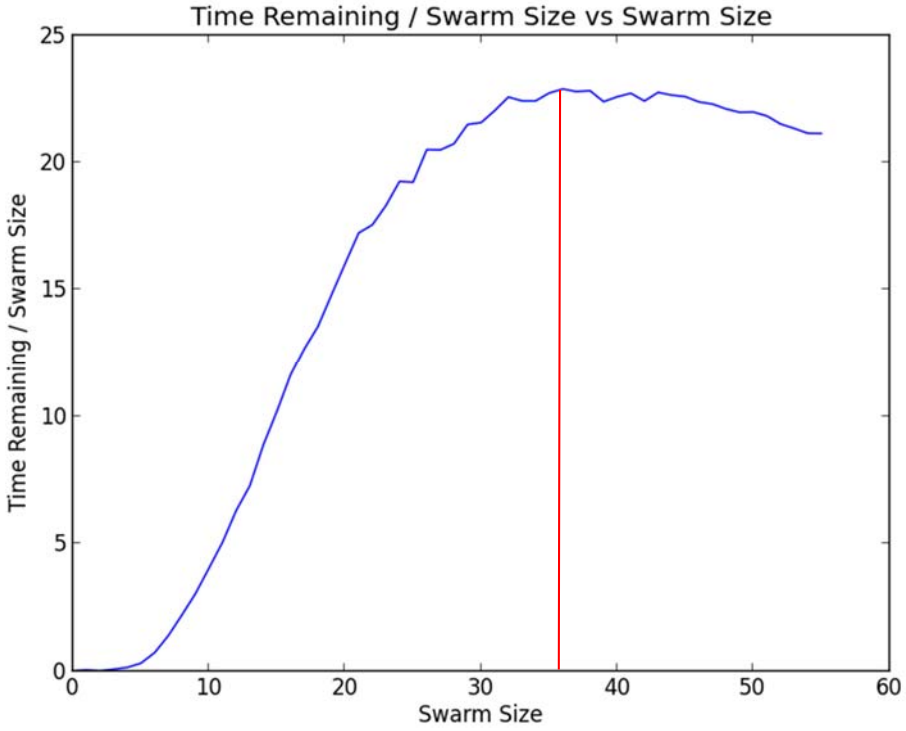

Figure 38.Time Remaining / Swarm Size. As shown here, the time remaining divided by the swarm size peaks at a swarm size of approximately 36.

From analyzing these data, it is evident that a population size of at least 15 is necessary for a successful swarm and the swarms begin experiencing diminishing returns at a population size of 35 agents. Therefore, the optimum swarm size for this particular Minefield application, as described, is approximately 35 agents. The solution is robust enough that slight variations in the swarm size do not greatly impact the success or efficiency of the emergent behaviors. This optimum swarm size is based on the difficulty of the scenario, which can easily be changed by adjusting a variety of factors. First, increasing the time limit would lower the difficulty greatly. Also, modifying the size of the search space, by either changing the actual size of the playing area, or the sensor ranges of the agents would result in either an easier or harder scenario, depending on whether the relative size of the search space is increased or decreased. In comparison, it would take approximately 213 stationary units with the designed sensor ranges of 25 units to completely cover the area required by the problem, which has a radius of 800 units. Similarly, a single unit could search the entire space in 213 time steps, assuming it had a global source of information guiding it to prevent it from returning to a previously searched location. Both of those examples are extreme cases, but they show how the Minefield scenario can be solved over a period of time with a much smaller population size than 213 and without the use of global information as needed by the single agent.

5.5 Conclusion

We have designed a simple model of a minefield and evolved a swarm that can successfully map out all of the mines' locations within a set time limit. In order to determine a "successful" swarm simulation, we set thresholds on three separate objectives. When those criteria were met, the swarm is optimized for speed. We also test

for the optimum swarm size needed in this simulation and find that a population size of 15 is required, while approximately 35 agents would be ideal. By simply adjusting some parameters, a user could set his or her own criteria for success, and then test to see if the required swarm size was feasible for their specific application.

CHAPTER SIX

Conclusion

Similar to ants in nature, the agents described in this thesis are not following complex algorithms. Instead they are being controlled individually by a handful of limited-range sensors and their corresponding weighting functions. The agents are able to perform their tasks without the use of a centralized controller. State-switching decisions are made at the agent level based on sensor readings in the local area and are modeled as threshold functions in most cases. The ability to switch between multiple tasks improves the performance of the entire swarm by adding robustness to its emergent behavior.

Throughout this research, multiple swarm scenarios are designed. The point-attack and point-defense simulation shows how a swarm can autonomously re-balance its resources between offense and defense in a battle scenario. Although this scenario is programmed to simulate swarm versus swarm situations, the task allocation methods found in the solution can be applied to multiple situations where a swarm needs to accomplish both offensive and defensive objectives. For the search and destroy mission, agents learn to spread out in order to search the area more effectively. Then, in order to destroy a much stronger enemy with the ability to fight back, the agents use recruitment methods to form into groups. The emergent behaviors of the previous two swarm simulations are then combined in the base attack scenario. A single swarm is able to keep a friendly base alive for a long period of time against enemy attacks by patrolling the area near the home base. At the same time, other agents spread out to search for

enemy targets and return to the home base for recruitment. As the number of defending agents decreases, the swarm compensates by having nearby explorers switch tasks to help defend the base. We also show that when the swarm loses a sensor, the evolutionary algorithm allows the swarm to compensate for that loss by either refining previous strategies, removing previous techniques, or forming all new methods. While in each case the swarm's overall fitness scores drop after losing a sensor, they are eventually able to maximize their scores by finding the best strategies given their remaining sensors. Finally, the minefield simulation demonstrates a strategy for a swarm to map out a group of underwater mines. The evolution method allows the user to select criteria by setting thresholds on multiple objectives. By fixing thresholds for objectives such as reporting mine locations and keeping as much of the swarm alive as possible, the evolutionary algorithm is then able to optimize the strategies for speed (or similar efficiency metrics). We also introduce a method for determining the optimal swarm size for a given scenario. This allows a user to specify the parameters of a scenario, set thresholds to define success and learn what the required swarm size for the solution is. The user can then compare the cost of the swarm intelligence solution with other methods to determine whether or not a swarm is the best method to accomplish their objectives.

In each case, an offline learning algorithm is used to evolve the swarms, maximizing their fitness scores by optimizing their parameters. In most cases, a complex fitness score is utilized to track the outcomes of the multiple objectives the swarms were working on. In some cases, such as the point-attack and point-defense swarm, fitness is computed by adding multiple factors, each weighted by the predefined importance of the objective. The base attack scenario is an example of another type of fitness function,

where teams being evolved are ranked lexicographically, based on how well they achieved a list of increasingly important objectives. In addition, a third option is used in the minefield application. In this case, the fitness score is computed as a product of four objective scores. Of course, the specific fitness function chosen depends entirely on the scenario tested, but in our cases, the third option of using a product for the fitness function seems to perform the best.

In conclusion, this research demonstrates the use of disjunctive control in multi-state swarms. Agents are able to switch states as needed by the swarm, allowing the swarm's behavior to be robust and flexible as the conditions of the simulation change. Inverting the swarm can be accomplished by evolving the parameters and is an excellent method for discovering sufficient rules for the swarm to follow that both meet the given criteria for success and keep open the possibility of unexpected results.

REFERENCES

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

2. 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

3. 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

4. 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.

5. Albert Yu, “Optimizing Multi-Agent Dynamics for Underwater Tactical Applications,” M.S. thesis, Dept. of Elec. and Comp. Eng., Baylor University, Waco, TX, 2011.

6. I. Gravagne and R.J. Marks II, “Emergent Behaviors of Protector, Refuge, and Aggressor Swarms,” IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol. 37, no. 2, pp. 471-476, Apr, 2007

7. D. Fogel, et al., “A self-learning evolutionary chess program,” Proceedings of the IEEE, vol. 92, no. 12, pp.1947,1954, Dec 2004

8. D. Fogel, Blondie24. San Francisco, CA: Morgan Kaufmann Publishers, 2002.

9. 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.

10. M. Dorigo, L.M. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” *Evolutionary Computation, IEEE Transactions on* , vol.1, no.1, pp.53,66, Apr 1997

11. Z. Yuan, “Continuous Optimization Algorithms for Tuning Real and Integer Parameters of Swarm Intelligence Algorithms,” ANTS 2010, pp. 203-214, 2010

12. 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. 58-72, Feb, 2002

13. D. Cvetkovic and I. Parmee, “Evolutionary Design and Multi-objective Optimisation,” Plymouth Engineering Design Centre, University of Plymouth. Drake Circus, Plymouth PL4 8AA, U.K.

14. F. Kursawe, “A Variant of Evolution Strategies for Vector Optimization,” University of Dortmund, Department of Computer Science XI, D 44221 Dortmund, Germany.

15. S. Carlson, “A General Method of Handling Constraints in Genetic Algorithms,” University of Virginia, Charlottesville, VA.

16. Jon Roach, R.J. Marks II, & Benjamin B. Thompson, “Tactical Task Allocation and Resource Management in Nonstationary Swarm Dynamics,” IJCNN 2013.

17. William E. Combs. Reconfiguring the fuzzy rule matrix for large time-critical applications. in 3rd Annu. Int. Conf. Fuzzy-Neural Applicat.,Syst., Tools, Nashua, NH, Nov. 1995, pp. 18:118:7.1

18. William E. Combs and J. E. Andrews. Combinatorial rule explosion eliminated by a fuzzy rule configuration. IEEE Transactions on Fuzzy Systems, vol. 6, no. 1, pp. 1-11, Feb. 1998.

19. Jeffrey J. Weinschenk, William E. Combs, Robert J. Marks II, "On the avoidance of rule explosion in fuzzy inference engines," International Journal of Information Technology and Intelligent Computing, vol.1, #4 (2007).

20. Sreeram Narayanan, R.J. Marks II , John L. Vian, J.J. Choi, M.A. El-Sharkawi & Benjamin B. Thompson, "Set Constraint Discovery: Missing Sensor Data Restoration Using Auto-Associative Regression Machines," Proceedings of the 2002 International Joint Conference on Neural Networks, 2002 IEEE World Congress on Computational Intelligence, May12-17, 2002, Honolulu, pp. 2872-2877.

21. Benjamin B. Thompson, Robert J. Marks II, and Mohamed A. El-Sharkawi "On the Contractive Nature of Autoencoders: Application to Missing Sensor Restoration," 2003 International Joint Conference on Neural Networks, July 20-24, 2003 , Portland , Oregon (pp. 3011-3016)

22. M.A. El-Sharkawi and R.J. Marks II, "Missing sensor restoration for system control and diagnosis," Symposium on Diagnostics for Electric Machines, Power Electronics and Drives, Atlanta, GA 24-26 August 2003, pp. 338-341.

23. V. Kumar, F. Sahin, “Foraging in ant colonies applied to the mine detection problem,” *Soft Computing in Industrial Applications, 2003. SMCia/03. Proceedings of the 2003 IEEE International Workshop on* , vol., no., pp.61,66,23-25 June 2003

24. E. Chapman, F. Sahin, “Application of swarm intelligence to the mine detection problem,” *Systems, Man and Cybernetics, 2004 IEEE International Conference on,* vol.6, no., pp.5429,5434 vol.6, 10-13 Oct. 2004

25. William J. Kirkwood, “AUV Technology and Application Basics,” *OCEANS 2008 – MTS/IEEE Kobe Techno-Ocean* , vol.,no.,pp.15,15,8-11 April 2008

26. B. Widrow; M.A. Lehr, “30 years of adaptive neural networks: perceptron, Madaline, and backpropagation,” *Proceedings of the IEEE* , vol.78, no.9, pp.1415,1442, Sep 1990