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

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

***Abstract*— Swarm intelligence, inspired by the group behavior of insects in nature, can be used to design control algorithms for groups of autonomous units in environments where a centralized controller is not feasible. Allowing agents within the swarm to take on one of multiple states allows the swarm greater flexibility and the ability to accomplish multiple objectives simultaneously. In this paper, a multi-state swarm is designed to simulate the mapping of a modeled underwater minefield. The rules controlling the movement of the agents are evolved in an offline learning algorithm, which is also used to optimize the state-switching behaviors in the swarm. In addition, the robustness of the solution is tested by simulation at various swarm sizes. The minimum swarm size necessary for completion of the mission is found, along with the point at which diminishing returns in the fitness of the swarm appears.**

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

I. Introduction

Swarm intelligence is inspired by the way that large groups of insects work together in nature [6]. An insect can often be modeled as following a set of simple rules, or instincts, while a large swarm of insects are able to coordinate and accomplish a complex objective, such as designing a beehive or building an anthill. One of the goals of our research is to apply some of the concepts of swarm intelligence to a large group of autonomous underwater vehicles (AUVs). These drones operate in an environment where radio waves do not travel well and communication is very difficult. This means a centralized controller cannot be used. Instead, we can use swarm intelligence to find a set of rules for the agents to follow which results in the desired emergent behavior.

An application of underwater swarms 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 [1][2]. 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.

II. Minefield

An underwater minefield mapping scenario is developed to explore possible strategies. In this scenario, a single swarm is tasked with searching an area for mines and reporting all the mine positions to the central base. 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. For the purposes of the simulation, we assume that the mine's location is broadcast out from the base to let the rest of the swarm know the location of previously mapped mines.

We also introduce a recharging state for the swarm. This concept of battery life is not considered in our previous work [3][4][5], but was something we wanted to add in order to approach a more realistic applicable solution. All units are initialized with a battery charge of 100. After each cycle, the battery charge 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. 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 a 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.

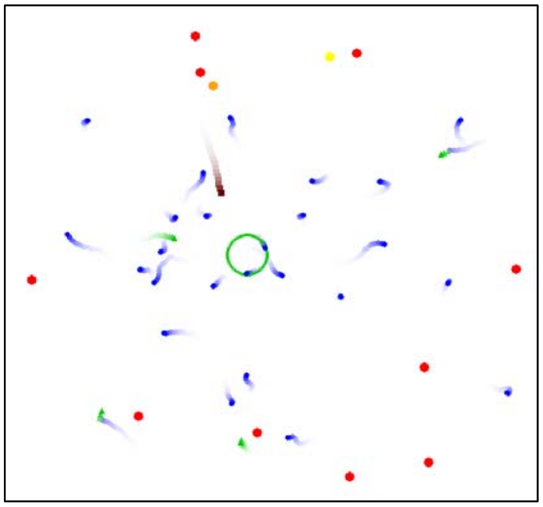

Figure 1. Screenshot from Minefield simulation. The small circles represent agents in the explore mode. The small triangles are agents in the recharging state and the small square agent is acting as a reporter. The larger dots indicate the location of the mines. The ring in the center of the map represents the sensor range of the central base (not pictured) as well as the recharging radius for the agents.

The movement of the agents is controlled by a series of actuator functions. Each agent is equipped with a variety of limited range sensors that can detect objects, such as other agents, mines, and the base. The distances to these objects are used as inputs for a series of actuator functions that determine the resulting movement with respect to the object being sensed. This combination of weighting functions is an example of disjunctive, or Combs, control [13]. In addition to the summation of the vectors generated from the group of actuator functions, each agent also takes a random step. The weighting functions themselves are simple piece-wise defined linear functions, as dictated by the y-values at three points. The y-values are set as adjustable parameters that can be modified to produce different behaviors. These parameters can then be optimized by using an evolutionary learning algorithm [8][9][10][11][12]. Agents are allowed to move off the edge of the map, but the swarm is penalized through their fitness scores if any agents are off the map at the end of the simulation. To prevent this, agents are also equipped with a center sensor that allows them to always sense how far away they are from the center of the map. If they move too far away, the center sensor's actuator function can be used to pull the agent back toward the center of the map. The point at which the agent are pulled back in is another evolved parameter.

Our optimization process begins by initializing a population of 80 swarms with random parameters. Then a Monte Carlo simulation of the entire population of swarms is performed. After simulation, the teams each receive a fitness score, described below, that defines how well the swarm performed in the previous round of simulations. The teams are then ranked by their fitness scores. Next, teams in the half of the population with lower fitness scores are removed from the population of possible solutions and replaced with copies of the better teams, along with a small amount of randomly generated solutions. The copies are then mutated by adding random Gaussian noise to the swarms' parameters. This concludes on cycle, or generation, of evolution. After hundreds of generations of evolution, poor solutions are filtered out and the algorithm can approach an optimum set of parameters.

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 successfully find at least 90% of the mines. It is important to note that 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 are 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 if 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:

$$Fitness = (1 + P_F)(1 + P_R)(1 + P_S)\left(1 + \frac{T}{2000}\right)(1)$$

In equation (1), P_F is the percent of mines found, P_R is the percent of mines reported, P_S is the percent of swarm remaining and T is the time remaining (out of the total time limit of 2000, 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.

III. Results

Figures 2-5 represent the progress of improving these objective percentages over the course of the evolution process. Figure 6 shows the resulting fitness scores over the same period of evolution.

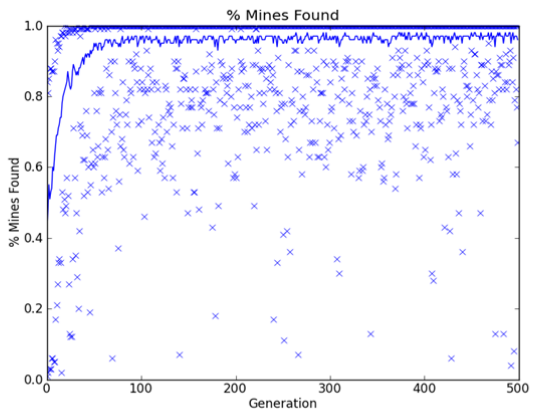

Figure 2. Mines Found. The average percentage of mines found reaches 97%, exceeding the goal of 95%. The maximum percentage found is 100%.

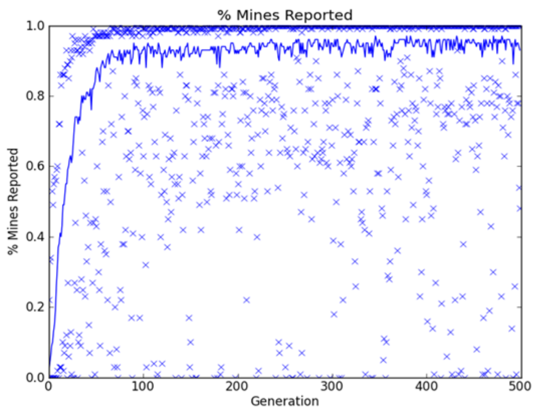

Figure 3. Mines Reported. The average percentage of mines reported for each generation is shown by the solid line. Maximum and minimum percentages for each generation are also indicated by x's. the average percent reported reaches the 95% threshold by the end of the evolution. The maximum scores approach 100% as

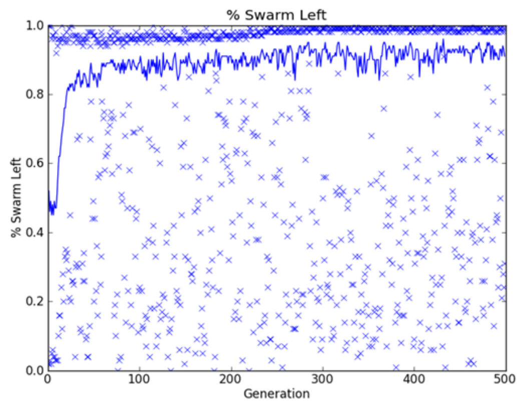

Figure 4. Swarm Left. On average, over 90% of the swarm survives the simulation by the end of the evolution process, which meets the requirement of 90% survival. Also, the maximum percentage of the swarm remaining reaches approximately 99%.

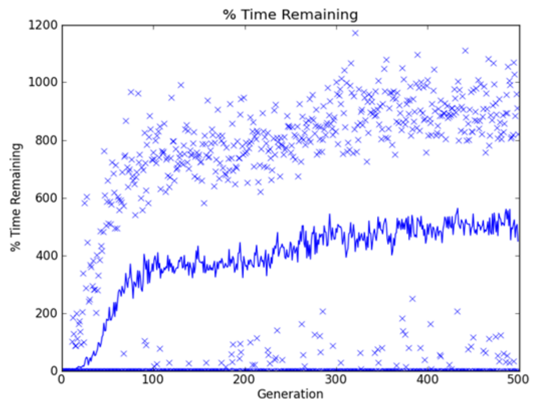

Figure 5. Time Remaining. The average time remaining at the end of a simulation converges to approximately 500. The maximum time remaining reaches 1000.

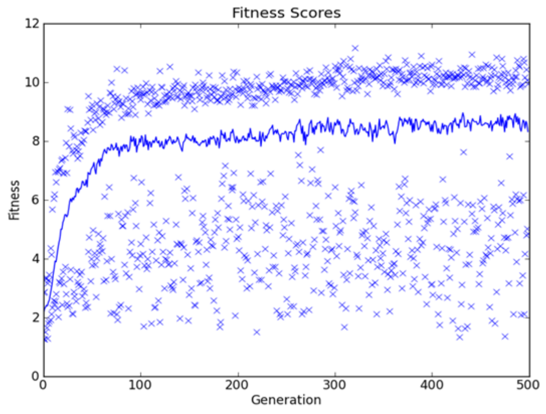

Figure 6. Fitness Scores. Average fitness scores settle at 8.5, with maximum scores greater than 10.

When this simulation was first designed, the goal was to produce an emergent behavior similar to the leaf-cutter ant [6]. 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, as all of the faster ants would get stuck behind slower ants.

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 [7]. 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.

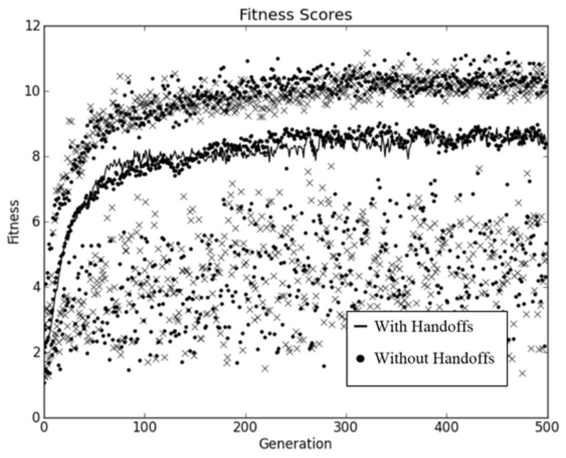

Figure 7. Comparison of Fitness Scores during evolution with and without handoffs. The red lines represent the scores generated by an evolution with handoffs. The blue show the fitness scores generated without handoffs. The results are essentially the same.

Unfortunately, it was discovered that this method of handing off bits of information does not improve the swarm's speed or efficiency in this 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 7, the results proved than 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, handoffs were shown to be a useless ability, and therefore may not be a necessary technological addition. The handoff capabilities can be removed from the simulation to simplify the rules without negatively affecting the emergent behaviors of the swarms.

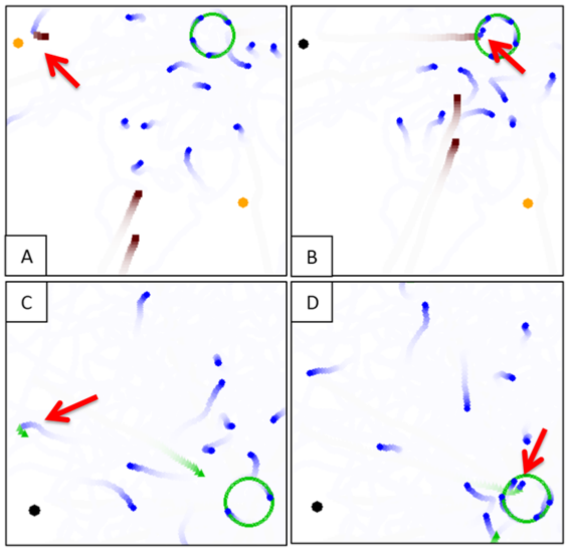

Figure 8. Examples of emergent behaviors. In A, a scout (circle) has just found an unreported mine. The scout switches to become a square reporter and returns to base. In B, the reporter reaches the base and reverts to scout mode. In C, a blue scout realizes its battery is too low and switches to a recharging state (triangle). In D, it has finished recharging at the base, and becomes a scout again.

Even though the solution did not achieve the method we were expecting, the swarms were 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 8.

IV. 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 many 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 outperform the benefits. Finding the optimum number of agents to successfully and efficiently accomplish an objective would be 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 was to design a swarm that would be large enough to have complex emergent behavior, but still small enough to be relatively inexpensive. We expected the fitness scores to increase as the population sizes increase. At some point, however, we anticipated seeing diminishing returns as we increase the swarm sizes. To test this, we took a highly evolved swarm solution and tested its performance when we changed the number of agents in the swarm. During the size test, the solution was tested 5000 times at each of the population sizes, which ranged from a swarm of one to fifty.

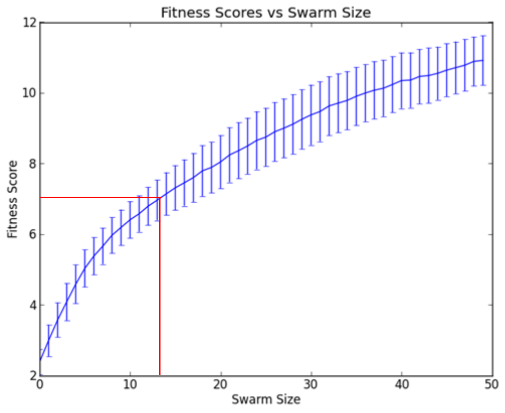

Figure 9. Fitness Scores vs Swarm Size. The average fitness scores for each swarm size, along with errorbars representing one standard deviation from the mean, are shown in this figure. A score of at least 7 represents a swarm that has successfully completed its objectives. Here, the average fitness scores reach this threshold at a size of 13.

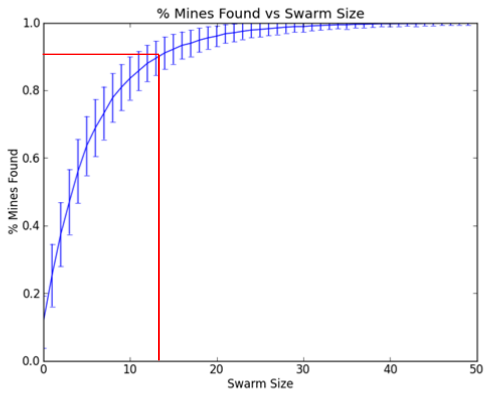

Figure 10. Mines Found. The percentage of mines found reaches the threshold of 90% at a swarm size of 13 agents, and begins experiencing noticeable diminishing returns around 36.

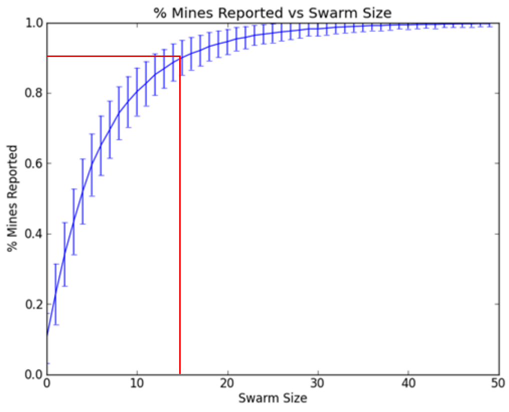

Figure 11. Mines Reported. Similar to Figure 10, a swarm of 15 agents is required to meet the threshold of the mines reported and again, diminished returns can be seen near 36 agents.

Figure 9 shows the increase of the fitness scores as the swarm size changes. As expected, we start to see diminishing returns and the swarm size gets unnecessarily large. Figures 10 and 11 show the percentages of mines found and reported. It can be seen that a swarm of fifteen 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 36 agents. After 36, there is not much difference in these two graphs. Figure 12 shows how the percentage of swarm 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 completed this objective and the mine search objectives. Another benefit of increasing the swarm size is shown in Figure 13, 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 36 agents and slows down after that. Figure 14 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.

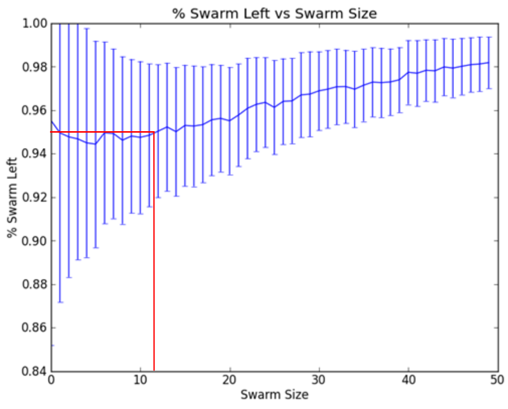

Figure 12. Swarm Left. 12 agents are required to meet, on average, the 95% survival criterion.

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 36 agents. Therefore, the optimum swarm size for this particular Minefield application, as described, is approximately 36 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 if you increased or decreased the relative size of the search space. In comparison, it would take 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.

Or, a single unit with could explore the entire search space in 213 time frames, 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.

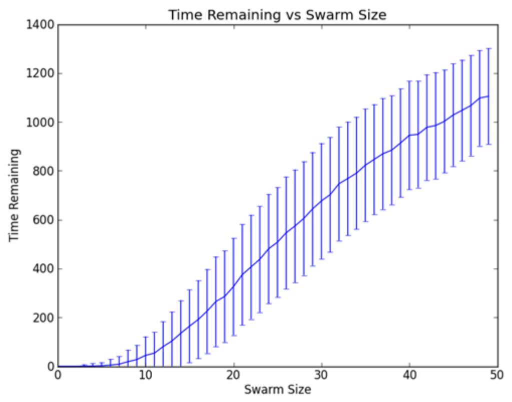

Figure 13. Time Remaining. The time remaining at the end of a successful search increases with swarm size.

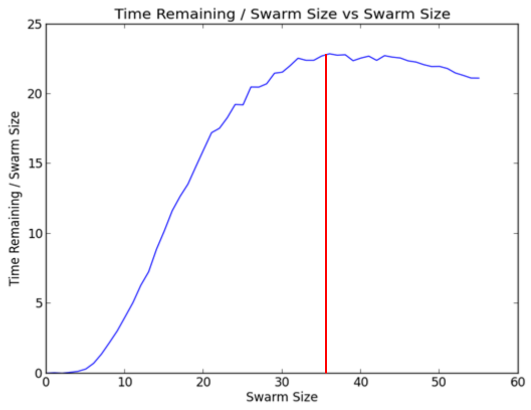

Figure 14. Time Remaining / Swarm Size. A shown here, the average time remaining divided by the swarm size peaks at a swarm size of approximately 36. This means that increasing the size of the swarm up to approximately 36 agents increases the speed of the swarm. After 36 agents, however, the addition of agents does not speed up the swarm enough to justify the cost of additional units.

V. 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 was optimized for speed. We also test for the optimum swarm size needed in this simulation and found that a population size of 15 is required while approximately 36 agents would be ideal. By simply adjusting some parameters, a user could set criteria for success, and then test to see if the required swarm size was feasible for their specific application.

Acknowledgment

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

References

[1] 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

[2] 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

[3] W. Ewert. R.J. Marks II, B.B. Thompson & Albert Yu, "Evolutionary Inversion of Swarm Emergence Using Disjunctive Combs Control," *IEEE Transactions on System, Man and Cybernetics* (2013).

[4] 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

[5] 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

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

[7] R. Reed & R.J. Marks II, Neuralsmithing (MIT Press, 1999)

[8] 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.

[9] Z. Yuan, "Continuous Optimization Algorithms for Tuning Real and Integer Parameters of Swarm Intelligence Algorithms," ANTS 2010, pp. 203-214, 2010

[10] 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

[11] 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.

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

[13] 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*