Jon H. Roach, Robert J. Marks II and Benjamin B. Thompson.
"Tactical task allocation and resource management in non-stationary swarm dynamics."
2013 International Joint Conference on Neural Networks (IJCNN), pp.1-5. DOI: 10.1109/IJCNN.2013.6706879

Tactical Task Allocation and Resource Management in Non-Stationary Swarm Dynamics

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

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

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

I. Introduction

An important component of swarm intelligence systems is division of labor. If there are multiple, possibly competing, objectives, how is a group of autonomous units able to decide how many units should work on each objective? In this paper, we will demonstrate the ability of these swarms to adapt to dynamic conditions by autonomously reallocating resources as necessary in order to achieve multiple objectives. Our solution is based on strategies found in nature, both the state-switching methods employed in ant colonies and recruitment techniques found in swarms of bees [1]. These methods are tested in a simulation that requires the swarms to accomplish two objectives at the same time: defending a friendly unit and attacking enemy targets. 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.

II. Swarm Intelligence

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

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

John H. Roach is with L3 Mission Integration, Greenville, TX. 75403 USA (e-mail: Jon_Roach@Baylor.edu)

Robert J. Marks is with the Department of Electrical and Computer Engineering, Baylor University, Waco, TX 76798 USA (e-mail: Robert_Marks@baylor.edu)

Benjamin B. Thompson is with the Applied Research Laboratory, The Pennsylvania State University, State College, PA 16804 USA (e-mail: bbt10@arl.psu.edu)

III. Development of Scenario

The evolved swarm had two objectives. First, the swarm needed to defend a central base from incoming projectiles. Agents could detonate themselves to destroy the enemy projectiles. However, these explosions would also take out friendly units nearby. Second, the swarm needed to seek out enemy units that were spawned at the edges of the playing area. These enemy units periodically fired projectiles towards the central base. Enemy units were programmed to be stronger and required three agents to detonate nearby within a short period of time. This requirement was added to force the swarms to form groups which would involve learning recruiting techniques. When the friendly base took too much damage and was destroyed, the simulation ended. A effective swarm in this simulation would be able to defend its base while simultaneously searching for and destroying enemy units. Swarms were scored on how long they survived and how many agents stayed 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 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 two sets of weighting functions to determine the resulting movement. One set of weighting functions determines movement toward or away from the object being sensed, while the second set controls movement parallel to the object. 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.

The only exception to this rule is a center sensor. Much of the unique behavior of the swarms typically occurs at the edges of the playing area. However, in many applications there may not be a hard boundary to the search area. So instead of a strict, rectangular boundary which was used before, we modified the game to use a softer, circular boundary. The units are able to go anywhere, but after they reach a certain distance from the center, they are inclined to move back in using a separate weighting function. For this sensor we made the distance at which the sensor was turned on an adjustable parameter. This allowed the swarm to actually learn an optimal area to search within. Since the center sensor was different for units in different states, this sensor actually resulted in some unexpected recruitment techniques which we will explain later.

A. Multiple States

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 and destroying incoming projectiles, while the attackers are capable of searching for the enemy units, forming groups and attacking the enemies in force. Part of the inspiration for this division of labor is found in colonies of ants. 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. Similarly, agents within the swarm are able to switch between offense and defense as needed.

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

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

B. Recruitment

In addition to deciding when to switch states, the agents also needed 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. For our simulation, the pheromone trail method does not work well with our application, so we decided to implement only the first method. Agents are able to sense which other nearby agents are asking for help and respond according to some preset rules.

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

IV. Evolution Process

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

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

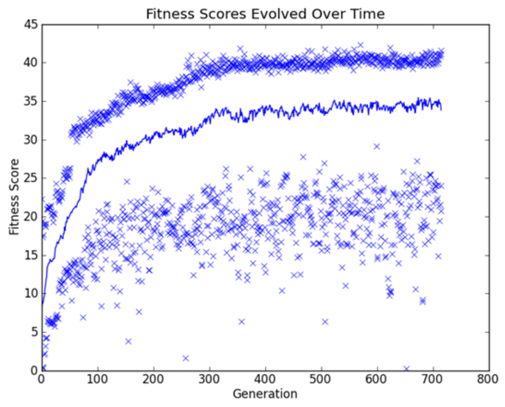

Figure 1. This figure represents the learning process of the evolutionary algorithm by showing how the fitness scores improved over time. After 300 generations, the algorithm converges to an optimal solution.

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

The new 25 teams are mutated by adding random Gaussian noise to the weights that control the swarms' behavior. This process allows the evolutionary algorithm to remove poor solutions and keep successful solutions, while constantly searching for new and improved strategies that are both similar to previous good solutions and different enough that the search is considering new strategies. The mutation step size is an important parameter in the evolutionary program. If the step size is too small, then the program will not be able to effectively search through the entire search space. On the other hand, if the step size is too large, then the search will not be able to converge to a solution. In order to prevent the search from converging too quickly, a minimum step size was used. The minimum step size was calculated by first calculating the average step size for each weight over all of the teams tested. After a list of the average step sizes was calculated, the minimum step size was found by selecting the median of the average step sizes using the method described by Liang et al. [8].

V. 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 searching for and destruction of enemy units. The improvement of the fitness scores over the course of the evolution process is shown in Figure 1. 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.

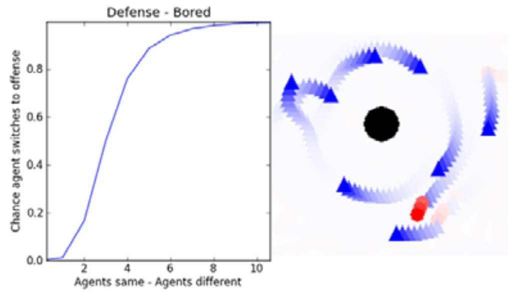

Figure 2. The threshold function shown here demonstrated the ability for agents to switch states if they decide there are too many units working on their task and are "bored". First, the agent counts up the number of nearby agents working on its task and those working on a different task. This is fed into the threshold function, which determines the chance that the agent will switch. In this case, if the difference between nearby defenders and attackers is 3, then the agent has a 50% chance of switching to offense. If the number of defenders compared to attackers is large, the agent will most likely switch, and vice versa. In this image, a blue defending agent has decided that there are too many defenders around it and chooses to switch states to become a red scout. (For a video of this swarm, see http://NeoSwarm.com/videos.html.)

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 2 and 3 demonstrate the dynamic state switching behaviors learned by the swarms.

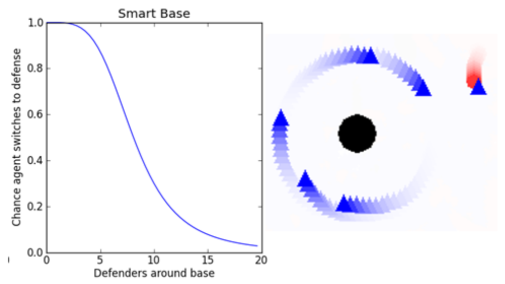

Figure 3. This function represents the way the agents process the information broadcast from the base. The base will broadcast the number of defenders around it and the agent has the option of switching to a defensive mode if it decides there are not enough defenders around it. In this case, the swarm will attempt to keep at least 8 or so defenders around the base. If the number of defenders is less than that, then nearby attackers will most likely switch to a defensive mode. Here, the base has broadcast that there are 6 defenders around it. A red scout has heard the message and decided that 6 defenders is not enough, so it chooses to switch states to become a blue defender. (For a video of this swarm, see http://NeoSwarm.com/videos.html.)

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. The defenders also learned to surround the base while maintaining a set distance from each other. They learned to keep their distance because detonations to destroy enemy projectiles could destroy friendly units if they were too close.

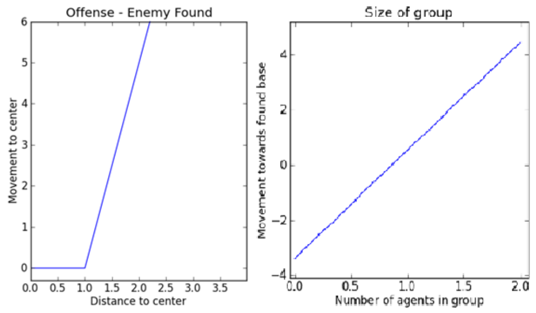

Figure 4. The first function shows the center sensor for recruiters. Note the scale for the x-axis. Any recruiters that are more than 1 unit away from the base will be inclined to return to the base. In other words, all recruiters return to the base. It is easy to understand why this unexpected strategy 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.

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 4 and 5.

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

VI. Conclusion

One of the advantages of swarm intelligence is a swarm's ability to autonomously reorganize itself in a dynamic environment. In our work, we have used techniques found in nature to allow a swarm to manifest this behavior in a simulation where the swarm is required to perform well in two objectives. The swarm has 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 were 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.

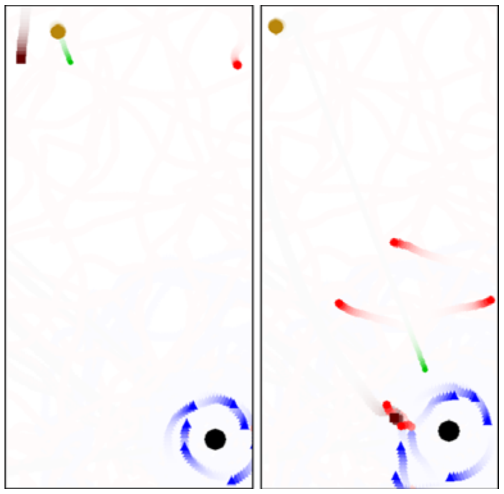

Figure 5. Here, a red scout has found an enemy unit and switched states to become a maroon recruiter. The recruiter is headed back toward the base in order to recruit other agents to form a group large enough to take out the enemy. This image shows the recruiter being joined by a third agent, which was 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. (For a video of this swarm, see http://NeoSwarm.com/videos.html.)

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

ACKNOWLEDGMENT

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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