Automatically Evolving a General Controller for Robot Swarms

John Ericksen
University of New Mexico
Department of Computer Science
johncarl@cs.unm.edu

Melanie Moses
University of New Mexico
Santa Fe Institute
melaniem@cs.unm.edu

Stephanie Forrest
Arizona State University
Santa Fe Institute
stephanie.forrest@asu.edu

***Abstract*—Controller design is an important problem for swarm robotics. Although many successful controllers have been proposed, most are hand-coded, sometimes using adaptive mechanisms to tune parameters of a manually designed algorithm. These solutions are generally tailored to specific environments, or problem instances, and often fail to scale well as swarm size is increased. This paper focuses on the problem of swarm foraging, proposing an automated method for designing scalable controllers that can perform effectively in multiple foraging environments. We use Neuroevolution of Augmented Topologies (NEAT) to design a neural network controller for a swarm of homogeneous robots. Our system, called NeatFA (NEAT Foraging Algorithm), is compared to existing swarm foraging algorithms, the Central Place Foraging Algorithm (CPFA), and the Distributed Deterministic Spiral Algorithm (DDSA). We find that NEAT produces controllers with performance that is comparable to both the CPFA and the DDSA. This is significant because the controller design was evolved automatically without preprogramming high-level behaviors or movements. The evolved neural network controller responds to sensed inputs and produces movements and actions that lead to effective collective foraging by the swarm. We find that the NeatFA controller performs comparably or outperforms the DDSA and CPFA for large swarm sizes. Finally, we show that a NeatFA general controller, when evolved for multiple environments but smaller swarm sizes, scales successfully to larger swarm sizes.**

I. Introduction

In swarm robotics, multiple autonomous individual robots work collectively to achieve a common goal. The use of inexpensive and redundant hardware in swarm robotics has clear advantages over the traditional single-robot approach for many problems[1][2]. A robot swarm can distribute work across many individuals, often distributed through physical space, allowing the swarm to explore more territory over time. Since the swarm is comprised of many redundant individuals, the swarm itself is more error tolerant than a single robot performing the same task, in the sense that it can lose individuals (e.g., to hardware failures) without affecting overall performance. This avoids the single-point-of-failure problem present in centrally controlled robotic systems.

The foraging problem is an instance of a common task for swarm robotics, namely, finding resources in potentially inhospitable or dangerous environments. For example, robots are expected to be used commonly in outer-space exploration, where expensive and costly life support is required for humans to survive[4]. The foraging problem applied to resource collection on a new planet may manifest itself as finding and collecting fuel or materials, often without a predefined map of the environment.

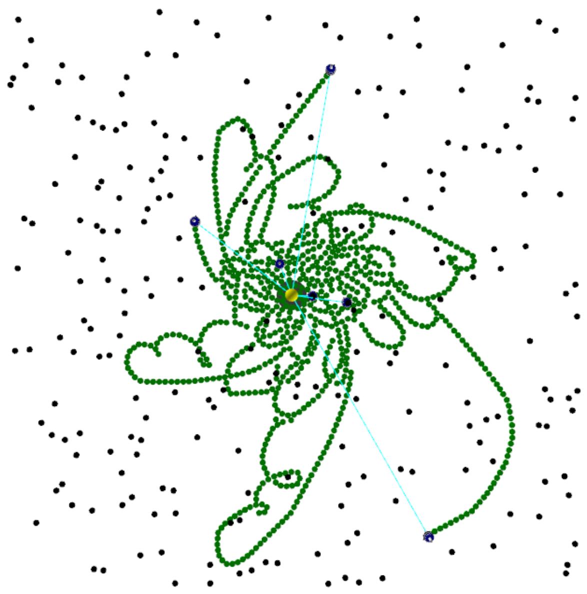
Fig. 1. Overhead visualization of NeatFA in ARGoS[3]. This image displays the random distribution swarm controller collecting seeds in the random environment. The green dots dispersed around the center of the arena represent pheromone placed by the robots in the swarm.

To solve the foraging problem, a robot swarm must collectively solve several sub-tasks[5][6]. First, the swarm must leave the base station (nest) and canvas the environment for resources, e.g., food. Once discovered, the robots in the swarm must pick up the food and find their way back to the nest and deposit it. After the individual robot has completed this foraging cycle, it then resumes the search for additional resources. Finally, multiple individuals in a swarm lead to interactions between robots, which can positively or negatively affect overall performance.

This paper describes how neural network controllers can be automatically generated for the swarm robotics foraging problem using Neuroevolution of Augmented Topologies (NEAT)[7]. First, we demonstrate that automatically designed controllers can perform acceptably on the foraging task. Then,

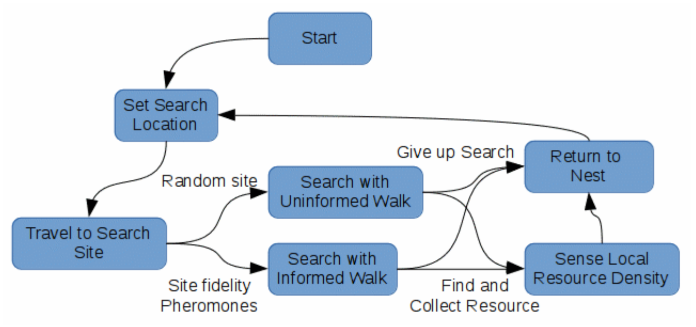

Fig. 2. The CPFA behavior state machine

we ask how well the automatically designed controller performs compared to other human-designed swarm controller implementations. Finally, we ask whether NeatFA can produce a swarm controller that generalizes—performing well in multiple different environments—addressing a significant limitation of today's swarm controller designs for foraging problems.

To investigate these questions, we compare the performance of a robot swarm controlled by a NEAT-designed neural network to two similar swarm controller algorithms; the Central Place Foraging Algorithm (CPFA)[8] and the Distributed Deterministic Spiral Algorithm (DDSA)[9]. The DDSA is entirely human-designed and implements a regular search pattern. The CPFA has human-designed behaviors that are governed by adaptively tuned parameters. Our algorithm, NeatFA, is automatically produced by NEAT, with the human only selecting the neural network inputs and outputs.

There are three major contributions of this paper. First, we demonstrate the ability to automatically evolve both environment-specific and general controllers using NEAT, which forage effectively in multiple environments. Second, for large swarm sizes, we show that our controller matches or outperforms both biologically inspired (CPFA) and hand designed (DDSA) controllers. Finally, we show that our general controller scales well to larger swarm sizes after being designed using much smaller swarm sizes.

A. Related Work

GAs are used in robotics for both physical and controller design, an approach known as *evolutionary robotics* (ER) [10]. Recent efforts in the physical domain of ER have focused on genetic algorithms. For example, designing "soft" body structures comprised of simple voxel blocks[11], rearrangeable and reconfigurable modular body parts[12][13], and evolutionary algorithmic design of rigid body parts[14][15]. These efforts center on robot mobility in different and challenging terrains. For control, GAs have been used to optimize parameters for configuring a controller[16]. Another approach is *novelty selection* [17] where the traditional fitness function is replaced with a function that rewards novelty—controllers that exhibit novel behaviors. By focusing on discovering innovative behaviors, the GA can potentially escape local minima within a behavior parameter set.

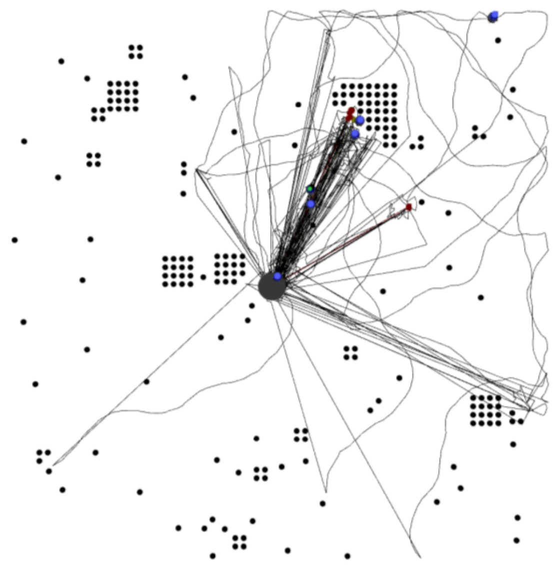

Fig. 3. Overhead visualization of the CPFA in ARGoS[3]. Fixtures in the environment include the gray nest at the middle, the blue dots as the robots, the black seeds in a semi-clustered configuration, and the line traces highlighting the paths of the robots.

GAs have often been used to evolve and train neural networks for various tasks [18][19][20][21][22][23][24][25]. In particular, they have been used to develop controllers for swarm robot foraging, as described by Timmis et al. in [26][27]. This work proposed a neural-endocrine system combined with a feed-forward back-propagation layered perceptron neural network for the swarm controller. The neural network within the neural-endocrine system is preprogrammed with specific tasks for the robot to execute. A limitation of this approach is the necessity of reprogramming or retraining the neural network whenever a new behavior is required. Our approach is quite different, using NEAT to design a cyclic perceptron network and setting the weights of the network prior to controller execution. This allows the neural network powering NeatFA to be automatically re-evolved for a given environment.

The CPFA[8] is a collection of hand-coded behaviors arranged in a state machine, as outlined in figure 2. Each state is governed by a set of parameters that determine movement, communication and memory. For example, parameters of the CPFA determine the probability of returning to the nest on any time step and the rate at which pheromones are deposited. Each robot in the CPFA has the same controller with identical parameters. These parameters are evolved before deployment to tailor behavior to a specific distribution of resources (seeds) and a particular swarm size.

The DDSA[9] is an algorithm designed to exhaustively search an arena for seeds following a non-overlapping square search pattern. The DDSA uses a recurrence relation that defines a preplanned path for each robot to collectively trace a spiral to fill the search space. When a robot encounters a target it returns it to the center and takes a direct path back

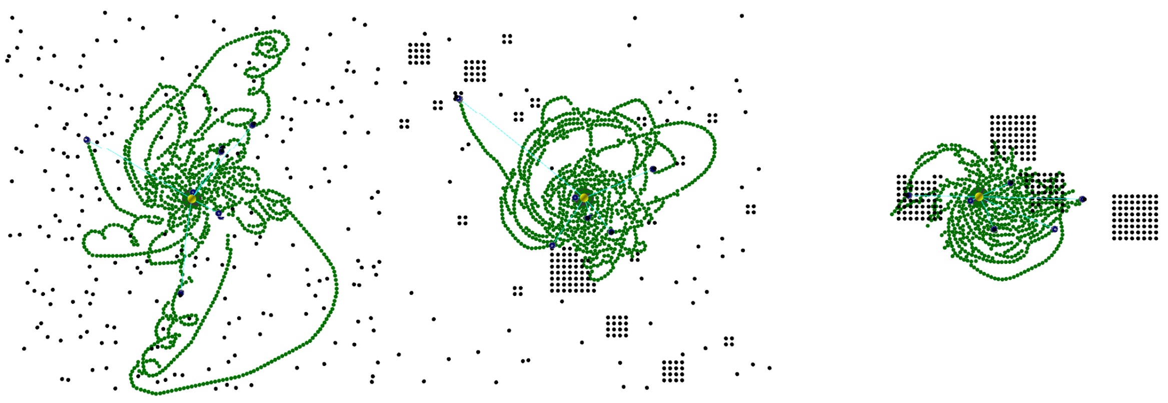

Fig. 4. NeatFA running in the three distributions; from right to left: random, semi-clustered, and clustered. In comparison to the CPFA and the DDSA visualizations, the NeatFA also contains a virtual nest, seeds and robots. Mirroring the functionality of the CPFA and the DDSA, a light is present above the nest allowing the robots to seek out and find the nest. The green dots in the environment are a visualization of the robot's pheromone trail.

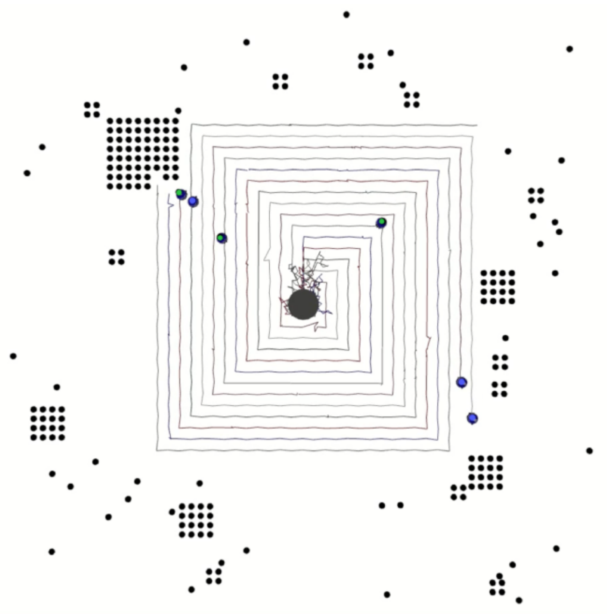

Fig. 5. Overhead visualization of the DDSA in ARGoS[3]. Similar to the CPFA featured in figure 3, the DDSA contains the same nest, seed, robot and paths. However, the DDSA's algorithmic difference is apparent in this visualization shown here as the square exhaustive search pattern.

to that location to resume its spiral search pattern. The DDSA was found to collect tags faster than the CPFA and it was surprisingly robust to localization error, but it was not scalable to large swarm sizes. The DDSA is visualized in figure 5.

Another evolved approach to the foraging problem is described by Ferrante et al. in [28]. This approach seeks to evolve parameters in a heterogeneous swarm to build a cooperative swarm with task specialization. By task specialization, it is shown that parameters are successfully evolved to specialize robots in the swarm into collectors and droppers. This approach differs from our approach as it allows for a heterogeneous swarm. For comparison sake, NeatFA uses a homogeneous controller architecture to match the CPFA and the DDSA algorithms.

The CPFA has been extended to support multiple nests in the Multi-Place Foraging Algorithm (MPFA)[29]. The MPFA follows a similar parameter evolution strategy as the CPFA, extended to multiple robot swarms that collaboratively forage in the same arena. This approach presents different collision avoidance issues related to the multiple drop off locations. Here we focus on single-swarm algorithms, leaving the MPFA for future extensions.

Neuroevolution of Augmented Topologies (NEAT)[7] is a GA that is specialized for neural network design. It searches for a good design by encoding a minimal network into a chromosome for the initial population, and then evolving the network using conservative network growth, feature-based crossover, and speciation. NEAT-constructed networks are comprised of a directed graph of connected perceptrons[30]. Each perceptron combines real inputs from both system sensors and other perceptrons to produce control signal outputs. The network of perceptrons acts as feed-forward universal approximator[31]. Here we experiment with the traditional version of NEAT, but other variants may be relevant for future work. HyperNEAT[32] [33] constructs Compositional Pattern Producing Networks (CPPN)[34]. The CPPN governs weight magnitudes on the connection edges in the neural network. Real-Time NEAT (rtNEAT) [35] evolves neural network controllers during the runtime of an experiment by replacing a fraction of the population periodically with more evolved individuals.

II. Methods

We implemented NeatFA to facilitate direct comparison to the CPFA and the DDSA. With that objective, NeatFA runs in ARGoS[3], so we can compare it to ARGoS simulations of the CPFA and the DDSA. ARGoS is a swarm simulation environment written in C++. It allows simulations to run either in high-performance 'headless' mode or in visualization mode. Headless mode is significantly faster than visualization mode. Both modes are critical, because GAs require repetitive simulations to evaluate multiple swarm's fitnesses over multiple epochs, but simulation visualization allows direct inspection of behaviors for analysis and verification, leading to insights into the implemented behavior.

The simulation environment includes a nest, a variable number of robot individuals and seeds available for foraging. The simulated arena is a 10 by 10 meter square grid. The nest is located in the center of the arena, 5 meters from the arena edges. Figures 1 and 4 give visualized examples of the simulation environment. Each simulation's trial duration is 30 minutes of simulation time. A trial starts with the robot individuals randomly distributed near the nest. Robot individuals in the simulation are moved by manipulating left and right wheel speeds. Experiments are run with 256 seeds placed in the arena in one of 3 distributions; random, semi-clustered and clustered. In the random distribution seeds are placed uniformly at random. The semi-clustered distribution places seeds in groups of 1, 4, 16 and 64 seeds, each arranged in square with each group placed randomly in the arena. The clustered distribution places 4 groups of 64 seeds, organized in an 8 by 8 seed grid, with each group located at a random location in the arena. In earlier work, the CPFA and the DDSA were analyzed on these exact distributions[8][9].

Our NeatFA implementation is designed to achieve similar functionality to the CPFA and the DDSA and makes the following assumptions:

1) As in the CPFA and the DDSA, the evolved controllers know where the nest is located in the arena. This aids navigation back to the nest. In NeatFA this is implemented by a light at the nest that is visible to the simple light-magnitude sensing eyes evenly distributed around the body of each individual robot.
2) All controllers can detect if their robot is holding a seed, and if so, the robot always returns the seed to the nest before continuing its search. NeatFA includes the holding-seed state as an input to the network, `+1` for holding a seed and `-1` for the absence of a seed. Similarly, an input is included that detects the presence of a seed at the robot's current location.
3) Both the CPFA and the DDSA algorithms are pre-programmed to pick up a seed when detected (if not already holding one) and to automatically drop a seed when it arrives at the next. NeatFA incorporates this preprogrammed behavior. This is the only prespecified behavior in NeatFA and is used to bootstrap the collection process.
4) Both the CPFA and the DDSA store state in the environment. In the CPFA, robots optionally lay pheromone trails that are available to itself and other individuals, mimicking the behavior of ants. The DDSA stores the spiral track information of each robot, allowing the robot to return to its previous location after delivering a seed to the nest. NeatFA allows the neural network to lay a pheromone drop if a designated output neuron has value greater than 0 and there is no pheromone within radius `R=5cm`. Further, the robot can detect the presence of a pheromone drop within radius `2R` (a `+1` value at a designated input neuron and `-1` for absence of pheromone).
5) In the CPFA, individuals can detect other individuals to facilitate collision avoidance. NeatFA implements this by including a set of range detectors evenly distributed around each robot's body. Range detector values are raw inputs to the NeatFA neural network.
6) Although the CPFA and the DDSA do not explicitly use the simulated compass heading inputs, the DDSA does follow vertical and horizontal paths while performing the spiral search. To mirror this functionality, current compass heading inputs are given as north, south, east, and west magnitudes as input to the NeatFA network.
7) The simulation built for the CPFA and the DDSA has four walls surrounding the arena to prevent robots from wandering to far away from the nest. The NeatFA simulation environment also uses walls.

NeatFA calculates fitness by awarding points to positive behaviors within the simulation: One point for picking up a seed and two points for returning it to the nest. This fitness strategy accumulates points for partially completed tasks to encourage swarms to complete all components seed foraging. The sum of these points over the run of the simulation across all robots in the swarm comprises the fitness for the associated chromosome. Each individual controller in the population is evaluated, and the fitness scores are reported to JNeat [36]. JNeat is a vanilla implementation of NEAT in Java, based on the original C++ implementation. Each evolution processing step produces a new population of robot swarm controllers to evaluate. Source code and supplemental materials for NeatFA is available on Github[1].

A. Experiment 1

The first experiment compares the performance of NeatFA to the CPFA and the DDSA in the semi-clustered environment evolved for specific swarm sizes. This experiment follows the approach of [9] which compared the performance of the DDSA to that of the CPFA. Each plotted point is the mean (error bars indicate the standard deviation) of 10 trial ARGoS simulation runs collecting seeds distributed in the semi-clustered distribution with swarm sizes of 2, 4, 6, 8, 10, 15, 20, 25, and 30. The previous result from [9] highlighted a problem with the DDSA: namely, swarm sizes greater than 15 suffer from congestion at the nest—inhibiting fitness. To compare against the CPFA and the DDSA algorithms, we evolved a population of 100 neural network controllers for each swarm size in the semi-clustered distribution over 300 generations. The swarm with maximum fitness was selected and then re-evaluated over the 10 trials. The final fitness of the neural network is divided by 3, which gives the number of seeds collected, and then divided by 256 to give the percentage of seeds collected.

[1]https://github.com/AdaptiveComputationLab/neatfa

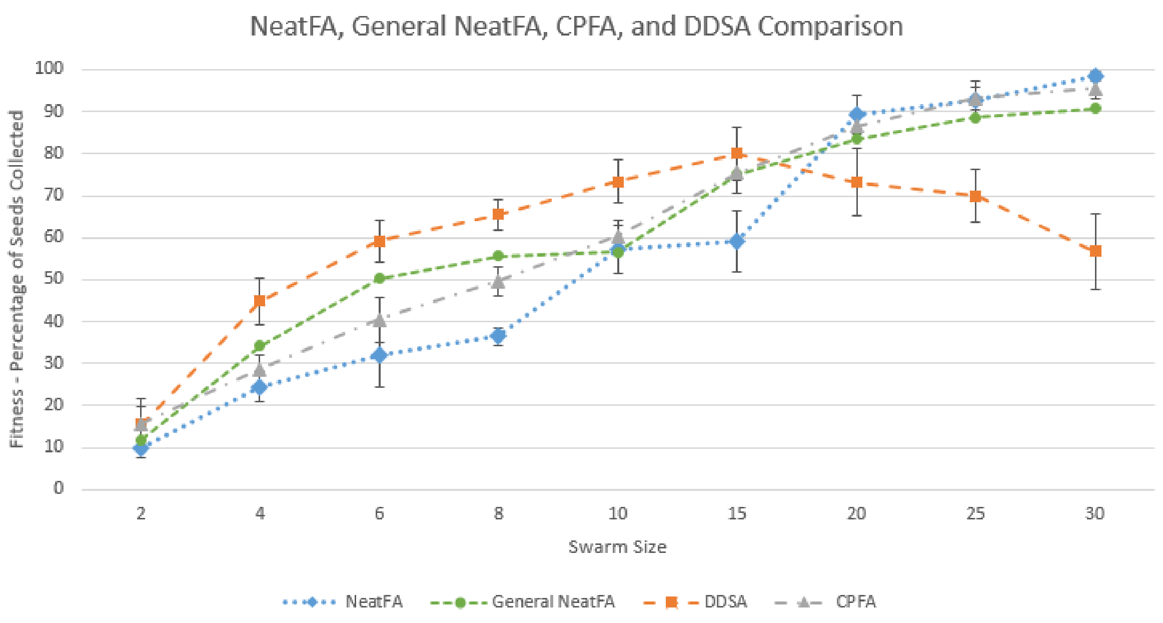

Fig. 6. Comparison of the CPFA, DDSA, NeatFA and General NeatFA in a simulation semi-clustered environment with a 30-minute simulation. Swarm sizes for the three algorithms are listed on the x-axis. Fitness is calculated as a percentage of total seeds collected out of the 256 seeds present in the simulation. Performance of the maximum fitness NeatFA general controller evaluated against the CPFA and the DDSA across the variety of different swarm sizes in the semi-clustered environment. NeatFA closely tracks the CPFA's performance and outperforms it for swarm sizes 4, 6, and 8. The general NeatFA outperforms the DDSA for swarm sizes larger than 15. Additionally, both NeatFA and General NeatFA do not suffer from the crowding issue apparent in the higher DDSA swarm sizes.

B. *Experiment 2*

The second experiment follows the approach of [8], in which the performance of parameter sets evolved for specific distributions were assessed on other distributions. This experiment evolves three separate NeatFA swarms, one for each distribution. We evolved a population of 100 individual swarms of size 6 over 300 generations in the 30-minute trial simulation against each of the 3 distributions. As before, the maximum fitness individual swarm selected from each of the 3 evolution runs is evaluated by finding the mean and standard deviation fitness by executing 10 trials across the 3 distributions each (evolved for random against random, clustered, semi-clustered; evolved for clustered against random, clustered, semi-clustered and so on).

C. *Experiment 3*

The third experiment seeks to find a general-purpose swarm algorithm with NeatFA. This experiment evolves a population of 100 swarms with swarm size of 6, calculating fitness in all three distributions. Each swarm controller is evaluated against 10 random, 10 semi-clustered, and 10 clustered environments. The 30 total distribution environments are configured identically for each robot swarm to ensure To avoid issues with lucky or favorable environment configurations, for instance, clustered distributions with the clusters near the nest, the same 30 distributions were used from swarm to swarm. In other words, the same 30 environment distributions are used for each swarm total fitness calculation through every epoch. The experiment is run for 300 epochs and the maximum fitness chromosome is selected from the experiment. The controller with the highest fitness score is evaluated against the 3 distributions and the swarm sizes targeted in experiment 1.

III. Results

A. *Experiment 1*

The results of evolving NeatFA for swarm sizes 2, 4, 6, 8 10, 15, 20, 25, and 30 are shown in figure 6 (plotted blue points). NeatFA performance ranges from 9.7% of the total seeds collected for a swarm size of 2 to 98.4% of the total seeds collected for a swarm size of 30.

The grey and orange plots show the performance of CPFA (grey) and DDSA (orange) in the semi-clustered environment with different swarm sizes (data reproduced from [9]), which highlights how DDSA performance falls off at large swarms sizes. As mentioned earlier this is caused by crowding at the nest, which interferes with swarm members dropping off seeds. Overall, the NeatFA performance on experiment 1 is similar to that of the CPFA, demonstrating that NeatFA avoids the nest crowding problem seen in DDSA, which we confirmed through visual inspection of the runs.

These experimental results also support the claim that the NeatFA performs comparably to CPFA and the DDSA, slightly outperforming both algorithms at higher swarm sizes. NeatFA's worst relative performance was at swarm size 8, where it collected 53.7% of the total seeds that DDSA collected, but at swarms size 30 NeatFA beat DDSA's performance by 173.6%. Compared to the CPFA, at swarm size

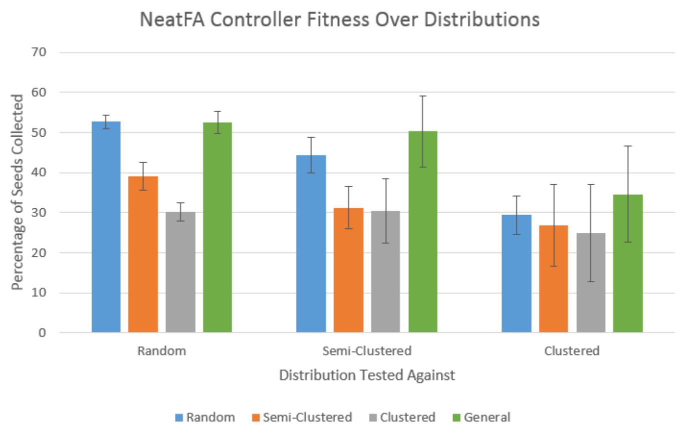

Fig. 7. Comparison of NeatFA evolved for the given distribution listed in the legend, executed against the distributions listed on the x-axis. Blue represents the performance of the controller evolved for the random, orange for the semi-clustered, and grey for the clustered distribution. The performance of the controller evolved for the random environment is noteworthy as it exceeds the fitness of the other two evolved strategies, even in their own distribution. The standard deviation highlighted by the error bars, shows that the random distribution controller had the lowest variance where as the clustered controller had the highest amount of variance. For comparison purposes, General NeatFA (described in Experiment 3) is shown in green. General NeatFA equals or outperforms the specifically evolved NeatFA in all distributions tested.

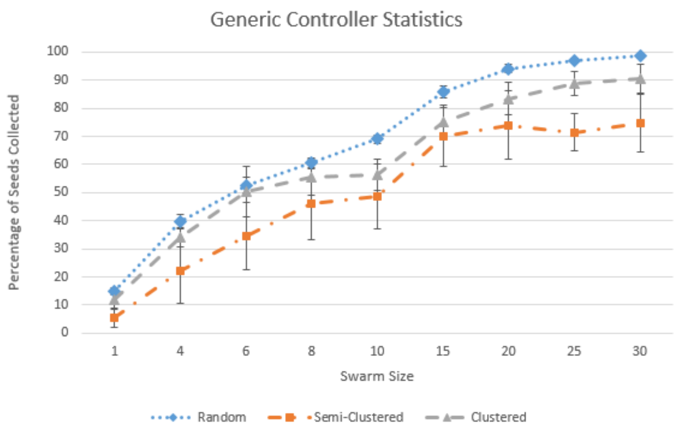

Fig. 8. Mean and standard deviation performance of the general NeatFA controller. The controller performs reasonably well in all three distributions and scales linearly across swarm sizes.

8 NeatFA collected only 61.9% as many seeds as CPFA, but at swarm size 30 it collected 103.2%. Both comparisons are within a factor of 2, signifying that the algorithms have comparable performance. More importantly, NeatFA achieved its results using an automatically designed algorithm.

B. *Experiment 2*

Experiment 2 results shown in figure 7 outlines the mean performance of the NeatFA evolved for the three specific distributions, evaluated against all three distributions with a swarm size of 6. Most noteworthy is the the swarm evolved for the random distribution, which outperformed the other controllers even those evolved specifically for the evaluated distribution. In contrast, the controllers evolved in clustered and semi-clustered environments fail to improve foraging in the environments they were evolved for. While the CPFA could be evolved in different resource environments to improve foraging in that environment, NeatFA did not evolve to strategies that improved foraging for clustered resources. Thus, we used a different approach in experiment 3 to design a generic controller that is sufficiently flexible to perform well in all 3 distributions.

The CPFA was shown to leverage pheromones more as distributions became more clustered. This begged the question, do pheromones benefit NeatFA? From visual inspection it was not clear that robots were following pheromones to clusters. Therefore, we disabled the ability to lay pheromone in each of the controllers generated in experiment 1. The resulting performance dramatically decreased to near zero — the robots were unable to effectively forage without pheromones. We conclude that the NeatFA controller relies on pheromones; however pheromone use does not improve foraging in clustered distributions as it does in the CPFA.

C. *Experiment 3*

Seeking a general controller that can perform well if it doesn't know the distribution of seeds ahead of time, experiment 3 evolves a controller for the three distributions simultaneously using a swarm size of 6. Figure 6 shows how the NeatFA controller performs as it scales up to the comparison swarm sizes. Its performance is comparable to the CPFA evolved for the semi-clustered distribution and specific swarm sizes. In other words, the NeatFA general controller scales well without requiring redesign for different swarm sizes. This controller also performs well in the clustered and random distributions as shown in Figure 8. The general NeatFA controller performs best in the random distribution across all tested swarm sizes with the semi-clustered and clustered distributions not lagging far behind. Figure 7 highlights the general NeatFA controller's performance across the three distributions in comparison to the specifically evolved NeatFA. General NeatFA is able to meet or outperform the specifically evolved NeatFA across the board.

The general NeatFA's performance does not decrease as swarm sizes increase as the DDSA does, indicating it does not suffer from nest crowding. The general controller is able to effectively avoid this pitfall without specifically evolving for the larger swarm sizes where this issue is present.

IV. Conclusion

NeatFA, an evolved neural network controller, is competitive with the best human designed foraging algorithms. The foraging problem is non-trivial and requires a controller to perform a series of subtasks including searching the environment, retrieving and delivering food to a predefined nest, and interacting with other robots in the swarm. In addition, foraging robots must avoid collisions with other robots in the swarm. Through the use of NEAT, NeatFA is able to automatically produce behavior that satisfies these requirements with a simple reward-based fitness function and basic preceptron neural network.

We have shown that the NeatFA controller is comparable to two other popular foraging algorithms, the CPFA and the DDSA, across several different swarm sizes and seed distributions. These results hold both for the NeatFA controller evolved for specific swarm sizes (experiment 1) and for the general controller evolved in experiment 3. NeatFA performs comparably to DDSA on small swarm sizes and avoids overcrowding at the next for swarm sizes 20 and greater. This is remarkable as the NeatFA controller design is only evolved to do so, avoiding costly and time consuming work of designing a controller by hand. The general NeatFA swarm controller performs well in all target distributions, and its performance scales linearly with swarm size. This result shows that NeatFA can discover behaviors that generalize across environments and scale with swarm size.

The NeatFA controllers were designed using a simple reward-based fitness function and a simple preceptron-based neural network. The controller neural network structure is remarkably simple, attributed to NEAT's slow network growth. For instance, the general controller includes only 50 total nodes (including 16 input and 3 output nodes) connected by 251 edges.

Taken together, these results demonstrate the feasibility of incorporating fully automatic design of controllers for swarm robotic foraging problems.

A future goal of this work is to run NeatFA on physical swarm robot hardware following the methods and experimental design in [8]. This is a challenge due to the reality gap between simulation and real world inputs and outputs. For instance, physical robot hardware may suffer from noise or faulty inputs in comparison to inputs from a physics engine. These differences will require analysis and inspection to allow an evolved controller to interact with the real world. We are hopeful that the flexible nature of a general algorithm will lend itself to bridging this reality gap and allowing NeatFA to run a physical robot swarm.

Acknowledgement

The authors thank the UNM Adaptive Computation Lab members Andrew Milligan, Cari Martinez, Cynthia Freeman, Jessica Jones, Joe Renzullo, and Padraic Cashin for their support and feedback on this research. Also, we thank the UNM Biological Computation Lab members Joshua Hecker, Matthew Fricke, and Wayne Just for their help gathering comparison data. We like to recognize Laura Patrizi for her contributions to the early stages of this research effort. SF acknowledges the partial support of NSF (1518878), DARPA (FA8750-15-C-0118), AFRL (FA8750-15-2-0075), the Sandia National Laboratories Academic Alliance, and the Santa Fe Institute. MEM was supported by NASA MUREP #NNX15AM14A and a James S. McDonnell Foundation Complex Systems Scholar Award.

References

[1] A. L. Christensen, R. OGrady, and M. Dorigo, "From fireflies to fault-tolerant swarms of robots," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 754–766, 2009.

[2] A. F. Winfield and J. Nembrini, "Safety in numbers: fault-tolerance in robot swarms," *International Journal of Modelling, Identification and Control*, vol. 1, no. 1, pp. 30–37, 2006.

[3] "Argos," http://www.argos-sim.info/, retrieved 2016-09-06.

[4] W. Fink, J. M. Dohm, M. A. Tarbell, T. M. Hare, and V. R. Baker, "Next-generation robotic planetary reconnaissance missions: a paradigm shift," *Planetary and Space Science*, vol. 53, no. 14, pp. 1419–1426, 2005.

[5] W. Liu, A. F. Winfield, J. Sa, J. Chen, and L. Dou, "Towards energy optimization: Emergent task allocation in a swarm of foraging robots," *Adaptive behavior*, vol. 15, no. 3, pp. 289–305, 2007.

[6] A. F. Winfield, "Foraging robots," 2009.

[7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[8] J. P. Hecker and M. E. Moses, "Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms," *Swarm Intelligence*, vol. 9, no. 1, pp. 43–70, 2015.

[9] G. M. Fricke, J. P. Hecker, A. D. Griego, L. T. Tran, and M. E. Moses, "A distributed deterministic spiral search algorithm for swarms," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 4430–4436.

[10] J. C. Bongard, "Evolutionary robotics," *Communications of the ACM*, vol. 56, no. 8, pp. 74–83, 2013.

[11] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 167–174.

[12] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[13] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3293–3298.

[14] J. E. Auerbach and J. C. Bongard, "Evolving complete robots with cppn-neat: the utility of recurrent connections," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 1475–1482.

[15] S. Luke and L. Spector, "Evolving teamwork and coordination with genetic programming," in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 150–156.

[16] P. J. Fleming and R. C. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control engineering practice*, vol. 10, no. 11, pp. 1223–1241, 2002.

[17] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty." in *ALIFE*, 2008, pp. 329–336.

[18] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex systems*, vol. 4, no. 4, pp. 461–476, 1990.

[19] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: many could be better than all," *Artificial intelligence*, vol. 137, no. 1-2, pp. 239–263, 2002.

[20] F. Gruau *et al.*, "Neural network synthesis using cellular encoding and the genetic algorithm." 1994.

[21] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms." in *IJCAI*, vol. 89, 1989, pp. 762–767.

[22] F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural networks*, vol. 14, no. 1, pp. 79–88, 2003.

[23] O. Witkowski and T. Ikegami, "Emergence of swarming behavior: foraging agents evolve collective motion based on signaling," *PloS one*, vol. 11, no. 4, p. e0152756, 2016.

[24] A. Acerbi, D. Marocco, and S. Nolfi, "Social facilitation on the development of foraging behaviors in a population of autonomous robots," *Advances in artificial life*, pp. 625–634, 2007.

[25] I. F. Pérez, A. Boumaza, and F. Charpillet, “Learning collaborative foraging in a swarm of robots using embodied evolution,” in *ECAL 2017–14th European Conference on Artificial Life*, 2017.

[26] J. Timmis, L. Murray, and M. Neal, “A neural-endocrine architecture for foraging in swarm robotic systems,” *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pp. 319–330, 2010.

[27] P. Vargas, R. Moioli, L. de Castro, J. Timmis, M. Neal, and F. Von Zuben, “Artificial homeostatic system: a novel approach,” *Advances in Artificial Life*, pp. 754–764, 2005.

[28] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo, and T. Wenseleers, “Evolution of self-organized task specialization in robot swarms,” *PLoS computational biology*, vol. 11, no. 8, p. e1004273, 2015.

[29] Q. Lu, J. P. Hecker, and M. E. Moses, “The mpfa: A multiple-place foraging algorithm for biologically-inspired robot swarms,” in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3815–3821.

[30] F. Rosenbaltt, “The perceptron–a perciving and recognizing automation,” Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep., 1957.

[31] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[32] J. Drchal, J. Koutník, and M. Snorek, “Hyperneat controlled robots learn how to drive on roads in simulated environment,” in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 1087–1092.

[33] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks,” *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[34] K. O. Stanley, “Compositional pattern producing networks: A novel abstraction of development,” *Genetic programming and evolvable machines*, vol. 8, no. 2, pp. 131–162, 2007.

[35] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, “Real-time neuroevolution in the nero video game,” *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.

[36] J. HomePahe, “Framework for neuroevolution (neat java),” *Available on¡ http://nn. cs. utexas. edu/softview. php*.