Evolutionary Design of Particles for Collectively Intelligent Problem Solving

Benjamin Bengfort, Kevin Harrison, and Philip Kim

Abstract—Self-organizing "flocks" of agents that display emergent coordinated behavior can be implemented using only local interaction rules. When agents are also given memory and a goal-based Finite State Machine (FSM) controller, the resulting team is able to solve a locate-and-collect task. We show that Evolutionary Computing techiques can be applied to the parameters that comprise the FSM states. Comparing the resulting FSMs to similar controllers described in the previous literature, we find that the evolved FSMs perform as well as or better than those designed by human observation. We close by suggesting methods for applying evolutionary techniques to the structure as well as to the numerical parameters of the controller, perhaps leading to the emergence of novel behavior.

1 Introduction

Ever since Reynolds demonstrated that flocks of birds could be simulated as multi-agent systems [16], wherein each agent's motion is determined solely by local interactions with other agents, there has been interest in extending such systems to solve more general problems. The self-organizing characteristic of these systems, whereby coordinated behavior emerges without a central control mechanism, suggests that even relatively simple decentralized systems might yield novel approaches to problems in both the computational and physical domains.

By abstracting the idea of "flocks" to n-dimensional spaces, for example, "particle swarms" [11], [7] or "cultural algorithms" [6] have been used for numerical optimization. In the physical domain, swarm behavior have been used multirobot team movement control [1], [4], [9], or during deployment of mobile robots or sensors. In [5], flocking behavior was used to maximize coverage during such deployments while still maintaining collective behavior. Other tasks being explored include urban pursuit [20] and robot herding of animals [18].

Many approaches augment the agents with such features as working memory [21], [10]. Rodriguez [17] added both memory and a finite state machine (FSM) controller that switched the agent between different sets of movement behaviors according to its local environment and current goal. He was able to demonstrate that a team of such agents was able to solve a resource locate-and-collect problem, and that a team programmed with flocking behaviors outperformed teams of agents that did not influence each other's movements.

Attempts have been made to formalize the design of such cooperative multi-agent systems [13], [3]. In this paper, however, we turn to evolutionary computation techniques. Genetic Programming (GP) [12], Evolutionary Strategies [15], and Genetic Algorithms [8] have demonstrated the potential of computational design techniques for machine learning, optimization, and even engineering. Finite state machines have been evolved to solve problems such as automatic target detection [2]. Evolving the FSM described by Rodriguez, therefore, seems like a natural extension of multi-agent problem-solving. This is not to be confused with evolutionary techniques that leverage swarms as in [19], [14].

In this paper, we have attempted to extend Rodriguez's findings in [17] by applying evolutionary techniques specifically to the agent control mechanisms. Rodriguez determined the structure of the FSM and the parameters defining each state through inspection and empir-

ical techniques. We attempt to determine the optimal parameters by an evolutionary process that runs a population of randomly-generated FSM configurations through a simulated problem and then evolves the population through fitness-based selection, recombination, and mutation genetic operators.

We will show that a novel finite state machine for movement control can be evolved using an evolutionary programming technique that performs as well as or better that those configured by a top-down design approach. We will present both the simulation using the best human designed parameters, then an evolutionary computation to evolve the individual particles in the simulation.

2 METHODOLOGY

In order to best demonstrate that an evolutionary process can generate an agent controller that is competitive with a human designed one, we have created an experimental setup that is intentionally similar to the work done in [17]. The task is for a team of simulated agents to search and gather resources that are spread across a large two-dimensional world with periodic boundary conditions, then return the resources to a designated home base.

The individuals of the team operate as a flock- local rules determining movement and goals lead to emergent flocking behavior. Each individual has a limited sight range and angle, the ability to communicate with other, nearby agents and a limited working memory. Particle behavior is governed by a finite state machine that determines how each particle moves given its individual state and the environment around it.

The task is competitive, as there are two teams in the environment competing for the resources and therefore the simulation is non-deterministic. Agents can mine resources not only from designated depots, but also steal from the home base of the other team. The winner of a simulation is determined as the team that has the most resources in their home base at the end of a predetermined number of time steps.

The evolutionary process uses this simulation to pit individuals created via genetic operations against the best designed human agent. The fitness of the evolved team is a function of the resources that they have collected vs. the resources the other team collected against them. The next generation of evolution is then dependent of the fitness of teams as run in the simulation. In the parlance of evolutionary computing, the genotype is the real valued vector that represents the finite state machine of the particle and the phenotype is the emergent behavior of the team as a whole as it participates in the simulation.

In this section we will present the details of the simulation, as well as the details of the evolutionary computation mechanism that searches for the optimal, most economic particle in the simulation.

2.1 Simulation

The movement behavior of particles in teams is governed by local interactions of particles within a particlar neighborhood. The particle updates its velocity at each timestep based on what it obeserves in the world around it. As a whole, this leads to self-organizing, emergent behavior of the flock as a combined whole of its individuals. In particular, movement behaviors are updated with six velocity components: cohesion, alignment, separation, seeking, clearance, and avoidance. Together these components enable flocking behavior, minimize collisions, and strike a balance between the exploration of the world and the exploitation of discovered minerals.

It is important to discuss each individual velocity component in detail, as these components are part of the evolutionary mechanism that we have employed. Every component has several parameters: the radius - the site distance and alpha, the site angle which determine the neighborhood of the particular component. Additionally the weight and priority of the component are used to compute the final velocity of the particle.

Cohesion: a vector pointing towards the average position of friendly agents in the neighborhood, with magnitude equal to zero when

the agent's distance to the average neighbor is zero and increasing quadratically with distance until it is equal to the maximum velocity when the distance to the average neighbor is r. The neighborhood includes all team members who are not guarding or stunned.

$$\vec{v_c} = v_{max}\frac{\Delta\vec{p}}{\|\Delta\vec{p}\|}\left(\frac{\|\Delta\vec{p}\|}{r}\right)^2$$

Alignment: a vector pointing in the same direction as the average velocity of friendly agents in the neighborhood, with magnitude equal to zero when the agent's distance to the neighbors' average position is zero and increasing quadratically with distance until it is equal to the maximum velocity when the distance to the average neighbor is r. The neighborhood includes all team members if they are seeking or spreading.

$$\vec{v_{al}} = v_{max}\left(\frac{\|\Delta\vec{p}\|}{r}\right)^2\frac{\Delta\vec{v}}{\|\Delta\vec{v}\|}$$

Separation: a vector pointing away from the average position of friendly agents in the neighborhood, with magnitude equal to the maximum velocity when the distance between the agent and the average position is zero, and decreasing quadratically with distance until it is zero when the distance to the average neighbor is r. The nighborhood includes all team members no matter their state.

$$\vec{v_d} = -v_{max}\frac{\Delta\vec{p}}{\|\Delta\vec{p}\|}\left(\frac{r-\|\Delta\vec{p}\|}{r}\right)^2$$

Seeking, Homing, and Mineral Cohesion: vectors pointing directly towards the target site with a magnitude proportional to the maximum velocity. These velocity components allow the particle to navigate ina meaningful manner. The target is stored in memory and this velocity component isn't influenced by nearby neighbors.

$$\vec{v_h} = v_{max}\frac{\vec{p_t}-\vec{p}}{\|\vec{p_t}-\vec{p}\|}$$

Clearance: a vector pointing in a direction orthogonal to the difference between the average neighbor position and the agent's position proportional the the maximum velocity. This component ensures that swarms have a wide field of view. The neighborhood of this component is team members in sight that are not guarding or stunned.

$$\vec{v_{cl}} = v_{max}\frac{\Delta\vec{p}_\perp}{\|\Delta\vec{p}_\perp\|}$$

Avoidance: a vector pointing away from an enemy agent in the neighborhood, with magnitude v_{max} when distance to the enemy is zero and decreasing linearly with distance until it is zero when the distance is r. Unlike all the other components, this can be applied multiple times each timestep–once for every enemy in the neighborhood.

$$\vec{v_{av}} = \sum_{i=0}^{n} v_{max}\left(\frac{r-\|\Delta\vec{p_e}\|}{r}\right)\frac{\Delta\vec{p_e}}{\|\Delta\vec{p_e}\|}$$

Also per Rodriguez, the FSM controlling each agent was always in one of four states: searching for new resources, moving to the last known resource deposit, carrying resources back to the home location, or guarding the home or a resource deposit. Each state was composed of some combination of the previously described velocity components, each characterized by a numerical weight and the radius and angle that defined the neighborhood.

Unlike Rodriguez, who prioritized each component and applied them in order, discarding any component that would have caused the velocity to exceed the maximum velocity, velocity in our simulation was a simple linear combination of the components; the maximum velocity constraint was only applied at the end to the resulting sum. Also unlike Rodriguez, agents in our simulation had "inertia" and started each timestep with the velocity of the previous timestep.

Transitions between states were triggered by conditions in each agent's local environment according to hard-coded rules. For example, an agent in the searching state that detected a mineral deposit within a 200-unit radius (in any direction) would push that deposit onto the top of its memory stack and transition to the seeking state (whcih is characterized by movement toward the top location on its stack).

One of Rodriguez's main findings was that it was advantageous for the agents to post

"guards" on their homes and/or on any discovered resource deposits. Agents that guarded only the home performed best of all, though agents that guarded both the home and deposits still bested non-guarding agents. In order for guarding behavior to be feasible, however, agents must avoid agents on the enemy team. Since we were allowing evolution to determine the strength of this avoidance parameter, we expected that in the absence of a "natural" motivation for avoidance, the avoidance parameter would be selected down to zero, allowing the agents to ignore enemy guards.

Therefore we implemented a penalty for colliding with another agent. An agent within 10 units of an enemy agent was rendered unable to move for a number of timesteps equal to 180 minus the angle of incidence in degrees. For example, if an agent collided with an opposing agent at right angles, the agent struck in the side would be unable to move for 90 timesteps, whereas the agent struck in the front would be unable to move for 180 timesteps. Similarly, being "rear-ended" by an opposing agent had no penalty at all. The hope was that by assigning the "responsible" agent more of the penalty, we would create an incentive to avoid collision.

In order to allow for the selection of guarding behavior, the transition to the guarding state was controlled by two evolvable parameters – the "home guarding threshold" and the "deposit guarding threshold" – that specified the number of friendly agents considered sufficient to guard the respective sites; a value of 0 disabled guarding behavior. This was the only instance in our simulation where a transition between states was controlled by an evolvable parameter.

At the beginning of each timestep, therefore, each agent is in a particular state. For each velocity component in its current state, it determines the number of friendly and enemy agents in the respective neighborhood and calculates the direction and magnitude of the component. These components are multiplied by the respective weight, added to the agent's velocity from the previous timestep, and the resulting velocity is subjected to the v_{max} constraint. The agent's position is updated by adding the velocity to the agent's position in the previous timestep. The agent then determines whether it should change its state based on the predefined transition rules and its current environment.

2.2 Evolution

After running each of the 50 configurations, we subjected the population to a round of evolution. The most fit individual from the current generation was always carried into the next generation, with the other 49 selected via tournaments of size 3. Each of the 49 was then subjected to mutation, with an independent 20% chance of mutating each weight, each radius, and each alpha of every component. If selected for mutation, the trait was changed by a random value uniformly-distributed between -0.2 and 0.2 for weights, -20 and 20 for radii, and -20 and 20 for alphas. Additionally, the home guarding and deposit guarding threshold each had a 20% of changing by plus or minus 1.

3 Experimental Procedure

Our computational experiment required two main components; a system to carry out the evolutionary computation, which constructed genotype populations via genetic operators, and a simulation system to instantiate phenotypes- particle swarms whose individuals are constructed from the genotypes. The simulation system computed the fitness of a particular swarm, which allowed the evolutionary computation to build the next generation.

The simulation environment consisted of two simulated teams in competition with each other; a red team and a black team. The simulated world was a 3000 x 3000 world with periodic boundary conditions and the number of timesteps was limited to 10,000 per simulation due to computational constraints. We restricted each team to 10 agents each,and evenly distributed 5 resource deposits betwen the red and black bases with 80 resources per deposit. Both the team size and the time limit were implemented to make the evolutionary computation tractable; our implementation required on average 273 seconds to complete; and each

generation took approximately 52 minutes to compute all fitness values.

The red team's behavior was implemented using a finite state machine, memory, and movement parameters exactly as the best empirical result designed by humans in [17], called *full guarding flock* meaning that the swarm guarded both its home base as well as discovered mineral deposits. The black team utilized the configuration derived from a genotype individual produced by the evolutionary computation. We attempted to hold as many variables constant as possible, including the locations of bases and resources, starting positions and velocities of agents, and update order. After 10,000 timesteps, the fitness of the black team was returned as the number of resources in the black base.

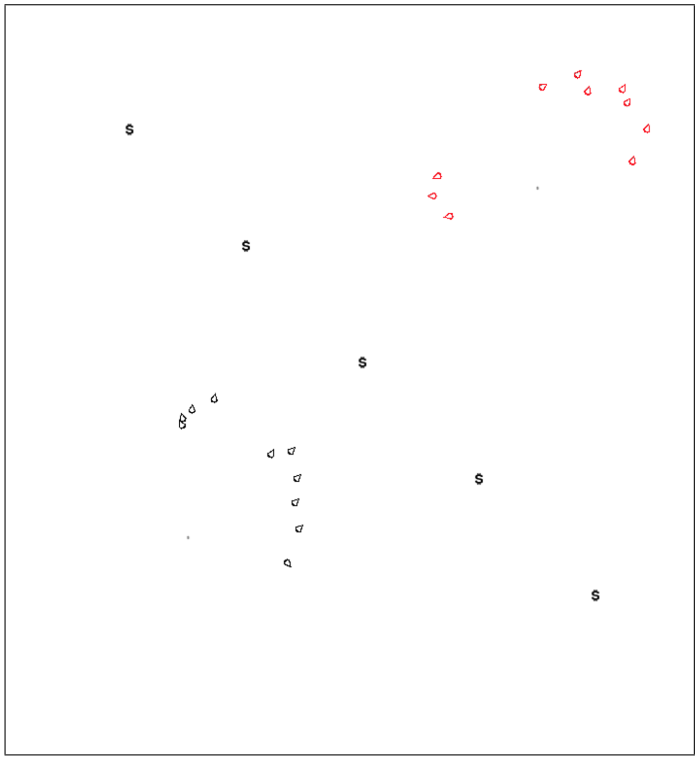

Fig. 1: Simulation of agents spreading in flocks.

Each simulation was run in parallel as subprocess of a master evolutionary computation process. The master process randomly initialized a population of 50 black team configurations, then ran a simulation for each individual in the population. After all 50 simulations were completed, tournament based selection with tournaments of size three were carried over to the next generation along with the 5 most elite individuals. After selection, genetic operators were applied.

In our first experiment we only used mutation to change the states in the finite state machine of the particles. In the second experiment we used recombination in addition to mutation operations. In either case, elite individuals were exempt from genetic operations, however elite individuals could be recombined to replace other individuals in the population. Both the mutation and recombination (when used) had a likelihood of 0.2. Maximum and minimum parameters were also placed on the values of the weight, the radii, and the alpha values to ensure an economic and reasonable result.

4 Results

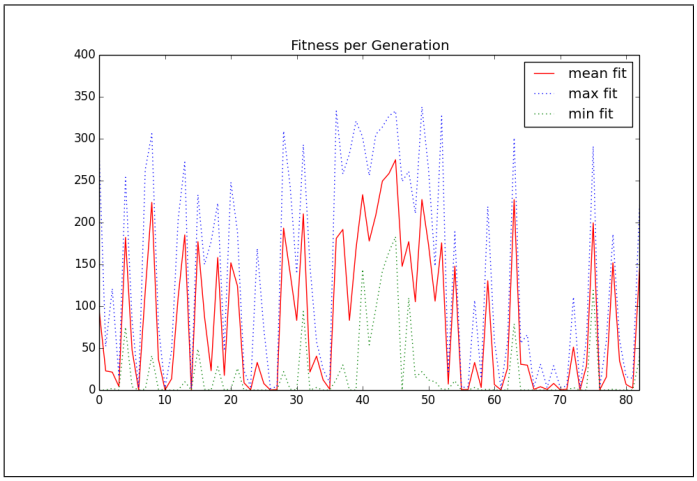

Fig. 2: Mean fitness per generation with mutation

5 Discussion

This work described herein represents only the first steps in the application of evolutionary techniques to this problem domain. By mutating and applying selective pressure to the movement component parameters, we were able to show that a machine could optimize the parameters better than a person. This was a form of optimization, but it did not result in novel behaviors. Indeed, since we hard-wired the transitions between the states (allowing for the guarding threshold to evolve) and allowed the evolver to mutate existing movement components in a given state but not to add them

(i.e., we pre-defined which components could have non-zero weight for each state), we held constant the ”meaning” of each state.

The next step would be to evolve the structure of the FSM itself; not only to allow the evolver to mix-in any component to any state, but to allow it to add new states, delete old states, and change the transitions between the states.

REFERENCES

[1] Tucker Balch and Ronald C Arkin. Behavior-based formation control for multirobot teams. *Robotics and Automation, IEEE Transactions on*, 14(6):926–939, 1998.

[2] Karl Benson. Evolving finite state machines with embedded genetic programming for automatic target detection. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 1543–1549. IEEE, 2000.

[3] Davy Capera, JeanYPierre Georgé, M-P Gleizes, and Pierre Glize. The amas theory for complex problem solving based on self-organizing cooperative agents. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 383–388. IEEE, 2003.

[4] Hande Çelikkanat and Erol Şahin. Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865, 2010.

[5] Ke Cheng, Yi Wang, and Prithviraj Dasgupta. Distributed area coverage using robot flocks. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 678–683. IEEE, 2009.

[6] Chan-Jin Chung and Robert G Reynolds. A testbed for solving optimization problems using cultural algorithms. In *Evolutionary programming*, pages 225–236, 1996.

[7] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[8] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[9] Jessica Hodgins and David Brogan. Robot herds: Group behaviors for systems with significant dynamics. In *Proceedings of Artificial Life IV*, pages 319–324, 1994.

[10] Xiaohui Hu, Russell C Eberhart, and Yuhui Shi. Particle swarm with extended memory for multiobjective optimization. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 193–197. IEEE, 2003.

[11] James Kennedy, Russell Eberhart, et al. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

[12] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[13] Maja J Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 432–441, 1993.

[14] Vladimiro Miranda. Evolutionary algorithms with particle swarm movements. In *Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference on*, pages 6–21. IEEE, 2005.

[15] Ingo Rechenberg. Evolution strategy: Natures way of optimization. In *Optimization: Methods and applications, possibilities and limitations*, pages 106–126. Springer, 1989.

[16] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[17] Alejandro Rodríguez and James A Reggia. Extending self-organizing particle systems to problem solving. *Artificial Life*, 10(4):379–395, 2004.

[18] Richard Vaughan, Neil Sumpter, Jane Henderson, Andy Frost, and Stephen Cameron. Robot control of animal flocks. In *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, pages 277–282. IEEE, 1998.

[19] Chengjian Wei, Zhenya He, Yifeng Zhang, and Wenjiang Pei. Swarm directions embedded in fast evolutionary programming. *networks*, 9:11, 2002.

[20] Ransom Winder and James A Reggia. Using distributed partial memories to improve self-organizing collective movements. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1697–1707, 2004.

[21] Ransom K Winder and James A Reggia. The role of working memory in an urban pursuit scenario. In *Artificial Life*, volume 13, pages 291–298, 2012.