Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2012
August 3-5, 2013, Portland, Oregon, USA

DRAFT PAPER – DETC2013-12485

EVOLUTIONARY DESIGN OF CELLULAR SELF-ORGANIZING SYSTEMS

James Humann
IMPACT Laboratory
Dept. of Aerospace & Mechanical Engineering
University of Southern California
Los Angeles, California 90089-1453
humann@usc.edu

Yan Jin*
IMPACT Laboratory
Dept. of Aerospace & Mechanical Engineering
University of Southern California
Los Angeles, California 90089-1453
yjin@usc.edu (**Corresponding author*)

ABSTRACT

In this paper, a genetic algorithm (GA) is used to discover interaction rules for a cellular self-organizing (CSO) system. The CSO system is a group of autonomous, independent agents which do not communicate with one another or any central controller. The agents have a local neighborhood of sensing and react only to other agents within this neighborhood. Their interaction rules are a simple set of direction vectors based on flocking rules. The five local interaction rules are assigned relative weights, and the agents self organize to display some emergent behavior at the system level.

The engineering challenge is to identify which sets of local rules will cause certain desired global behaviors. The global required behaviors of the system, such as flocking or exploration, are translated into a fitness function that can be evaluated at the end of a multi-agent simulation run. The GA works by tuning the relative weights of the local interaction rules so that the desired global behavior emerges, judged by the fitness function. The GA approach is shown to be successful in tuning the weights of these interaction rules on simulated CSO systems, and, in some cases, the GA actually evolved qualitatively different local interaction "strategies" that displayed equivalent emergent capabilities.

INTRODUCTION

Engineered systems are becoming increasingly complex. Current engineering design follows a reductionist approach. Reductionism can be defined colloquially as "divide and conquer." A large task is subdivided into several smaller tasks, and each of these tasks is completed and optimized somewhat independently. This approach is described at length in many fundamental and respected design textbooks. It can be seen in the function structure diagrams of Systematic Design [1], and the Independence Axiom in Axiomatic Design [2]. Many informal design methods, often company-specific, also follow the top-down reductionist approach.

The reductionist approach has worked well for many applications. It will continue to work well in these areas, and this paper does not argue otherwise. However, two of the fundamental assumptions in reductionist design are that the environment of the product is knowable and static (or possibly dynamic within a foreseeable range), and that the functional requirements (FR's) of the system will not change with time. Dealing with the complexity inherent in unknowable or dynamic environments and requirements presents unique challenges to the designer, which may be better solved using an alternate design method.

Self-organizing systems generally are distributed systems whose agents have limited capabilities and only local knowledge. Also, the overall distinctive behavior of the system is emergent; that is, it arises from the interactions of the simple agents and cannot be easily inferred from the properties of the agents [3][4]. Self-organization is present in a diverse range of system types, including physical, biological, and social systems. The hierarchical structure of materials can be thought of as a self-organization of atoms and molecules. "Hive mind" behavior in insects arises from simple agents with local interactions [5]. Even human social systems such as freeway traffic or evacuation exhibit emergent flocking behavior that can be re-created in simulation using self-organizing principles [6]. These complex structures and useful behaviors emerge on a system level with no central control or global knowledge on the local level.

A CSO (cellular self-organizing) system is an engineered system built bottom-up from a distributed group of agents known as mechanical cells (mCells). An overview of the CSO framework is given in [7], where the authors introduced a design DNA (dDNA) that governs the behavior of an individual cell. The designer's job is then to design the local behavioral rules of each cell in order to generate the desired emergent global behavior. One strategy suggested for this design work is field-based regulatory behavior where meaningful objects in the mCells' environment have "fields" surrounding them that the cells can sense and react to [8].

This bio-inspired design approach aims to develop emergent functionality from the local interactions of mCells. The promise of such a distributed self-organizing approach is that it can display some of the desired properties found in biological self-organizing systems, such as adaptability to environmental change, flexibility for changing functional requirements, and resilience to system damage. These desirable traits will be found in the system, even though the mCell hardware is assumed to be simple.

This simplicity is the foundation of the self-organizing approach. An mCell is assumed to have limited communication and computational abilities, so that mCells can be manufactured cheaply in large batches. The redundant nature of the system will provide robustness to damage to several mCells, as long as the damaged cells represent only a small portion of the system, and the self-organizing nature of the system will allow it to adapt to environmental conditions that were not pre-specified by the designer. Finally, flexibility will be gained by the ability of a designer to change the local rules of an mCell at the software level, so that different systems can display different emergent functionality, even if they consist of the same hardware.

Like any complex system, CSO systems have certain characteristics that make them difficult to design. Since their behavior is emergent, it is not always clear what the local rules of interaction should be. A designer must develop these rules using a combination of intuition, modeling, and what little established theory exists. There is a current need for automated methods of developing local CSO rules that can expedite and formalize this process.

In this research, we take an evolutionary approach to automating the design of a CSO system developed in our previous work [9], [10]. A genetic algorithm (GA) is developed to evolve self-organizing rules or parameters that govern the behavioral interactions among agents and therefore the emergent behavior of the system. In the following, we first review the related work and then introduce our evolutionary approach to CSO system design. After that, the experimental results are presented and discussed. This work is still ongoing, and the intermediate conclusions and future work are discussed in the last section.

RELATED WORK

Our research is related to multi-agent systems, especially the flocking model of multi-agent systems, self-organizing systems, genetic and evolutionary algorithms, and the application of GA to the optimization of agent-based systems.

Multi-Agent Simulations

Multi-agent simulations (MAS) are built to model phenomena that cannot be captured by simple mathematical formulas. There is no direct mapping from the parameters used to initialize an MAS to the resulting global behavior. Therefore, the only way to determine the results of a model is to actually run the model. In the case of complicated simulations, running the simulation could take a significant amount of computing time, and for a designer, the amount of design time required is the time per simulation multiplied by the number of simulations that must be run before suitable emergent behavior is found. If a system relies on more than a few parameters, and the proper combinations are found in small isolated portions of the search space, it becomes necessary for a designer to use some sort of efficient, automated search method [11].

Flocking

Reynolds [12] is credited with creating the first simple flocking algorithm. His motivation was to reduce the amount of time necessary for a computer animator to prescribe every path for every bird in a flock. Instead, he relied on independent agents, following a combination of three simple rules:

1. Collision Avoidance
2. Velocity Matching
3. Centering

where "Centering" is an agent's desire to stay near the center of the flock. From these simple local behaviors, an emergent behavior instantly recognizable as flocking could be animated with relative ease.

Artificial Self-Organizing Systems

While many researchers have studied self-organization in order to glean insight into biology or mathematics, some have purposefully integrated self-organizing principles into mechanical (or simulated) systems. Beckers et al. [13] created a system of several independent robots without communication that were able to gather pucks into a single pile. This approach was later expanded in [14] to require clustering in the center of the field, and using square objects. Werfel and Nagpal [15] have simulated a robotic construction crew which can build block structures. Zoein et al. [7] demonstrated a CSO systems which was capable of reorganizing its shape so that it could maneuver through or around various obstacles. Chiang [16] used variants of the flocking approach to demonstrate various emergent system behaviors such as spreading, clustering, searching and surrounding. Chen and Jin [8] demonstrated a simulated swarm of agents whose local interaction could be harnessed to push a box toward a goal while avoiding obstacles. Thus far, these systems have shown limited but promising capability, and improvement is difficult because of the indirect link between local actions and global behavior.

Genetic Algorithms

A genetic algorithm [17] is a stochastic optimization algorithm that uses DNA, natural selection, and evolution as a useful metaphor for efficient optimization. GA's are well-suited to explore discontinuous optimization search spaces. Because a

GA acts on a population of candidate solutions, and not just a single point, they have an ability to explore possibly remote areas of a search space in parallel while avoiding local maxima.

A GA works by generating an initial field of candidate solutions, or genomes. These genomes are typically binary strings. The genomes are mapped to specific arguments of a fitness function, and the fitness function evaluates each candidate's worth. Candidates with high fitness are more likely to be mated with other candidates, so that their offspring, composed of genes from both parents, survive to the next generation. Mutations can be introduced during the algorithm by randomly flipping a bit in a genome. This process is repeated for a set number of generations or until a candidate with suitable fitness is identified.

There are many different specific implementations of GA, each with its own strengths and weaknesses. Some hew closely to the Darwinian inspiration of the original algorithms, while others take liberties with the form of the genome, the number of parents involved in reproduction, cloning, etc. The important elements found in almost all GA's are selection (fitness), recombination (mating), and mutation.

Machine Learning for Tuning of Complex System Parameters

Many complex systems have been studied and optimized using genetic algorithms or other optimization methods. The methods used include GA, its close relative genetic programming (GP), linear programming and more.

Evolving complex systems using AI presents several challenges to the researcher. The architecture underlying most multi-agent simulation platforms is partly stochastic, even if there are no specific random parameters programmed into the simulation. This stochasticity can arise from random initial agent placements, stepping orders or initial conditions. The indirect global-to-local mapping makes it difficult to assign rewards for specific actions in reinforcement-based learning [18]. Also, the global behavior that one wishes to capture may be transient, emergent, or difficult to measure quantitatively [19]. Despite these difficulties, there has been a large amount of success in recent years in discovering desirable emergent behavior using optimization techniques.

Wilensky [20] specifically explored flocking behavior and how to evolve it using a GA. He successfully evolved flying-v behavior in a flock of simulated birds by allowing the GA to control the birds' cone of vision.

Ant foraging is a popular MAS that several researchers have attempted to optimize. Calvez and Hutzler [11] introduce an approach they call Adaptive Dichotomic Optimization. This approach uses parallel sampling and search space discretization to efficiently explore the search space of local behavior parameters in an ant foraging simulation. Cole [21] used a GA to optimize an ant colony for the shortest transit time across the colony by turning on and off various interaction rules and adjusting colony size.

GP has been used to implement firefly synchronization and ant foraging algorithms [22]. This work developed a string of functional primitives that had been used in previous algorithms to produce synchronization in simulated firefly illumination.

Miller [23] used GA to evolve parameters for a highly nonlinear model of world population dynamics known as World3. Here he demonstrated the use of GA for not only optimization of the final state, but also for sensitivity analysis. In order to study the sensitivity analysis, the fitness function for maximizing world population was modified by penalizing large changes from a set of baseline parameters. Thus only the parameters that provided the highest return for least amount of change survived the modified fitness function.

Crutchfield et al. [24] used GA to evolve rules for cellular automata to perform the "majority classification" task. The point of the task is for the cellular automata to converge to a steady state of all 1's if the initial condition is majority 1's, or all 0's if the initial condition is majority 0's. The first generations of the GA created simple rules that expanded groups of contiguous 1's or 0's, but later generations showed some "cleverness" in creating rules that caused structures to be built in the CA that could communicate local conditions to other locations on the CA.

A multi-objective GA was used in [25] to optimize a strategy in an emergency planning simulation. This algorithm used Pareto optimality to sort candidate solutions and then applied the typical GA operators to optimize disaster response plans for such targets as low fatalities and hospital waiting times.

Whether the objective was model optimization or data-fitting, the common theme among the works cited in this section is that they succeeded in adjusting more variables than a human mind could comfortably cope with simultaneously. The techniques used in the literature vary, as researchers have used diverse methods from GA to hill-climbing to simulated annealing, but the results show an efficient search of a vast, nonlinear space.

EVOLVING CSO SYSTEMS

The CSO approach has demonstrated some simple capabilities thus far, such as flocking, goal finding [16], system reconfiguration [7], and box pushing [8]. We wish to expand the functionality of the CSO system, without drastically changing the assumptions made about the limited capabilities of individual mCells, by training it using machine learning rather than human trial and error.

We propose to use genetic algorithms to explore the space of possible behavioral parameters in our CSO. By modeling the CSO as a multi-agent system, a multi-agent simulation can be used to analyze the global behavior emerging from the mCells' local interactions. The parameters controlling this local behavior can be coded as a numeric string to be manipulated by a GA. These candidate numerical strings can be efficiently probed by the GA to lead to the discovery of successful CSO strategies in a reasonable amount of time.

NetLogo [26] is a free multi-agent simulation environment that is well suited to study distributed systems with emergent behavior. NetLogo has an optional API controller that can be run on a Java virtual machine. A simple `command()` function in the Java console can send arguments to the NetLogo software as if it were a human typing into the NetLogo console. In this way, all system parameters can be set, and the simulations can be called by a Java GA program with no further input from the human operator.

Agent Behavior

The simulation used is a flocking simulation inspired by the work of [9] and [27]. The flocking agents represent small robots moving on a 2-D surface in a torroidal (wrapped) world. Each robot has a limited sensory range and very little memory. At each time step, a robot will sense the positions and heading of all the other robots within its radius of vision and react according to a set of predefined rules. These rules take the form of 5 vectors, which give competing desired directions. The relative weights of the vectors are assigned to the agents before the simulation begins, and the direction taken at any given step is the weighted sum of all five vectors. The 5 robot behaviors are defined as [16]:

1. **C**ohesion: step toward the center of mass of neighboring agents.
2. Av**o**idance: step away from agents that are too close.
3. **A**lignment: step in the direction that neighboring agents are headed.
4. **R**andomness: step in a random direction.
5. **M**omentum: step in the same direction as the last time-step.

The acronym COARM is used to refer to the weights assigned to each behavior. A robot calculates the cohesion, alignment, and avoidance directions according to the following formulas:

$$\vec{C} = \frac{1}{N}\sum_{i\in\eta}\vec{x}_i \qquad \textbf{(1)}$$

$$\vec{O} = -\frac{1}{N}\sum_{i\in\eta}\frac{\vec{x}_i}{\|x_i\| - D} \qquad \textbf{(2)}$$

$$\vec{A} = \frac{1}{N}\sum_{i\in\eta}\frac{\vec{v}_i}{\|v_i\|} \qquad \textbf{(3)}$$

where $i \in \eta$ implies that agent i is in the neighborhood of the agent calculating its direction, $\vec{x}_i$ is the vector from an agent to its neighbor, D is the diameter of the agents, and $\vec{v}_i$ is the velocity of an agent.

The denominator in (2) is used to ensure there are no collisions among agents. Since the vectors $\vec{x}_i$ are measured between the centers of agents, the avoidance term will increase toward infinity if the distance between the centers of any two agents approaches the agents' diameter. A maximum step size of 0.7 is enforced at any time step.

Because the behavior of the system relied predominantly on the ratios among the COARM parameters, rather than their absolute values, we decided to fix one value throughout the simulations. The value for R was fixed at 1, and all of the other parameters were allowed to adjust relative to this baseline.

Desired Emergent Behaviors

We wish to use a GA to evolve sets of parameters that lead to the following emergent flock behaviors:

1. *Flocking:* at the end of the simulation, all agents should have the same heading.
2. *Exploration:* at the end of the simulation, the agents should have discovered every location on the field.

We also slightly modify these goals to further challenge the GA and evolve other useful behaviors. For the flocking simulations, we introduce constraints on parameters to eliminate easy solutions. For the exploration simulations, we occasionally require that the system find only a certain fraction of the field – no more and no less. We map each of these global behaviors to a fitness function, and allow the GA to evolve sets of corresponding parameters.

Genetic Algorithm Specifications

The genetic algorithm is a custom program written in Java that controls a NetLogo simulation using the API. A string of 40 bits (5 binary numbers) is interpreted as the weights of the five COARM parameters. The GA assigns the parameters, initializes the simulation, gathers the results, applies the GA operators, and begins the next generation. The actual simulation and agent behavior take place in the NetLogo software.

Most genetic algorithms were seeded with an initial field of 15 candidates and allowed to run for 40 generations. Elitism was used so that the best candidate at each generation was cloned to the next generation. At each generation, fitness scaling [28] was used so that the selection probability of the best candidate was 30% greater than the selection probability of the average candidate. Two candidates were selected, with replacement, and their genomes were combined at a random cross point to create an offspring. All non-clones were allowed to mutate, with a mutation probability of 1% per bit of genome. This process was repeated until the next generation was full.

RESULTS AND ANALYSIS

Flocking

To evolve flocking behavior, an empty field was filled with 30 agents with random initial coordinates and headings. The COARM parameters were set by the GA, and the simulation was allowed to run for 250 time-steps. At the end of the simulation, the momentum of the entire flock $\vec{M}$ and total fitness were calculated assuming unit mass for each agent:

$$\vec{M} = \sum_{i=1}^{N} \vec{v}_i \qquad (4)$$

$$fitness = \frac{\|\vec{M}\|}{N} \qquad (5)$$

where N is the total number of flockers.

The fitness will be 1 if every agent is travelling in the same direction, and will have an expected value of 0 if the agents are heading in random directions.

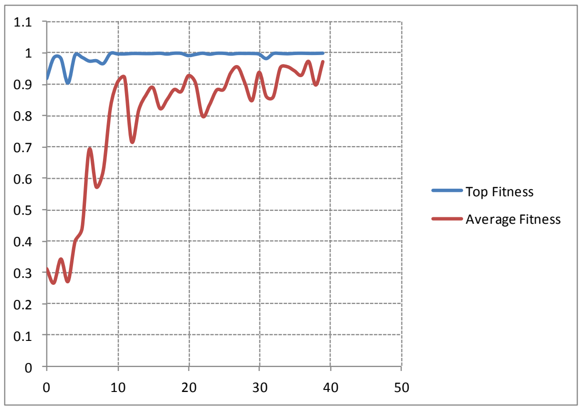

Figure 1: Typical flocking fitness evolution

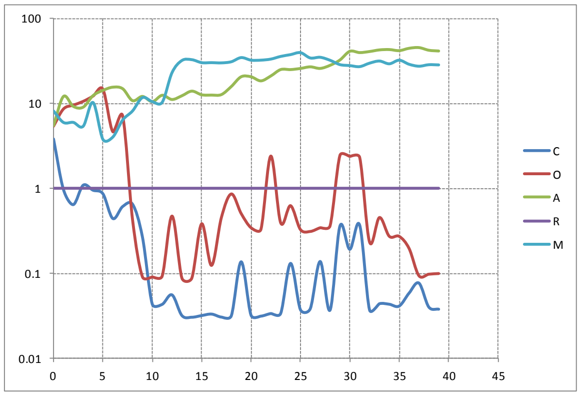

Figure 2: Typical flocking average COARM parameter evolution

This run of the GA, which allowed unrestricted manipulation of the COARM parameters, displayed intuitive results. As one would expect, the Alignment parameter (Figure 2) was quickly maximized so that all agents would have an overwhelming tendency to match their heading to their neighbors. Momentum was also maximized, which smoothed out any randomness, leading to a more consistent flocking direction, and Cohesion and Avoidance generally tracked one another, with the C/O ratio varying from 0.2 to 0.5, which allowed the agents to keep a suitable separation distance from one another – not too close to avoid collisions and flock segmentation, and not too far to avoid flock dispersion. Figure 3 shows the initial and final configurations for the MAS based on this parameter set. Other GA runs produced results that were qualitatively similar in relative parameter values and MAS results.

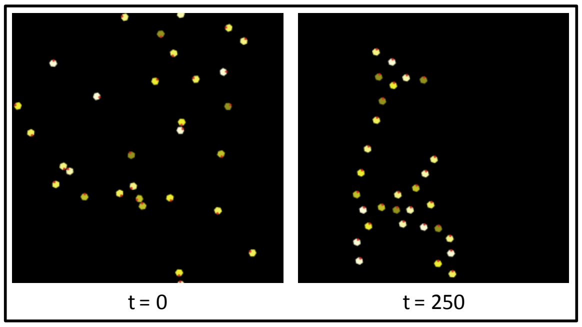

Figure 3: Self organization to form a cohesive flock

A "Hall of Fame" to Assess the Return on the GA Investment

It is often seen that a suitable set of parameters is found in the first few generations of the GA. Is it then wise to spend more computational effort on evolving further? To answer this question we instituted a "Hall of Fame," (HoF) which would house the best candidate from each generation. At the end of the GA run, each member of the HoF would be re-tested 5 times to calculate a minimum, average, and maximum fitness. This could be used to determine if higher quality, more reliable parameter values would appear in later generations, or if the randomly found best candidate in the first generation was sufficient.

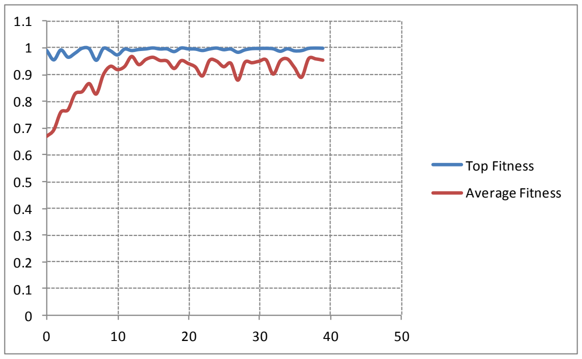

Figure 4: Fitness evolution with high-quality candidate in first generation

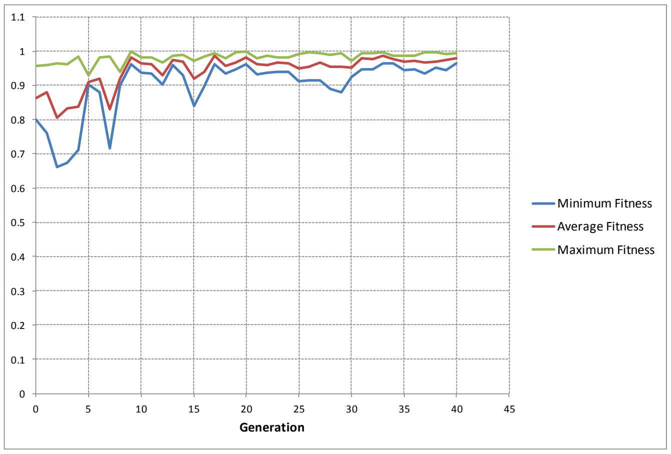

Figure 5: Each generation's best candidate retested for reliability

Figure 4 shows the fitness evolution in a GA that found a high-fitness candidate in the first generation by random luck. It is evident that there is not much improvement in the performance of each generation's best candidate throughout the simulation, but is there another way that we can sense improvement in succeeding generations? The simulation run is partly stochastic. There is an explicitly random vector, and the mechanics of the simulation environment are such that minor random variations can be introduced in the stepping order of the agents. Therefore, we expect that the fitness of any one candidate will be subject to a certain amount of noise, and that good candidates should not only display high fitness, but display high fitness reliably.

Figure 5 shows the results of the HoF from the same GA run. Note that despite the stochastic fluctuations in the middle generations, there is a clear improvement from beginning to end in the running of this GA, and that the worst performance of the final generation's top candidate is better than the best performance of the first generation's top candidate.

This shows that the GA was implicitly selecting for reliability all along, even though the candidates were only re-tested at the end of the simulation. It would be possible to test for reliability at each generation, but the time to complete one GA run would grow linearly with the number of repetitions required – see [29] and [30] for a good discussion of the tradeoffs in using repeated fitness evaluations for noisy fitness functions – and unnecessarily slow down the algorithm.

Because clones and near clones of candidates were being carried over into future generations, the reliability testing was essentially performed across generations, and the success is shown in the HoF results which test for reliability within generations.

Flocking with Parameter Constraints

Applying the GA to simple flocking may have been "too easy" for the algorithm, so we decided to test the algorithm with a new constraint, $A = R$; that is, the Alignment and Random factors will be equal. This is because, for the case of flocking, A and R counteract one another. In the original COARM formulation, R was meant to model system limitations such as noise in sensors or error in motor output [16], so we use it here to constrain a designer. Any relative importance placed on Alignment (which in practice would require more costly and precise sensors and actuators) will be offset with a Randomness penalty (representing the reduced performance of the rest of the robot).

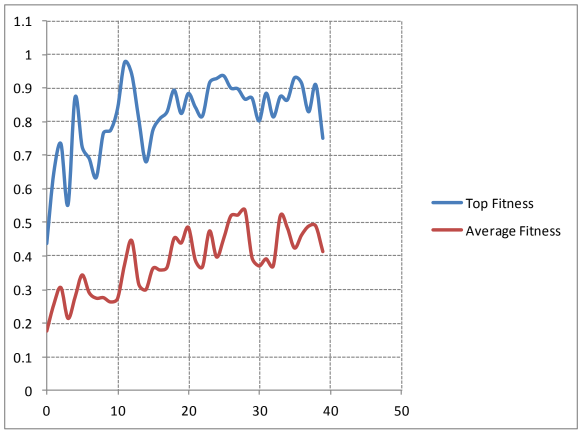

Figure 6: A = R Fitness Evolution

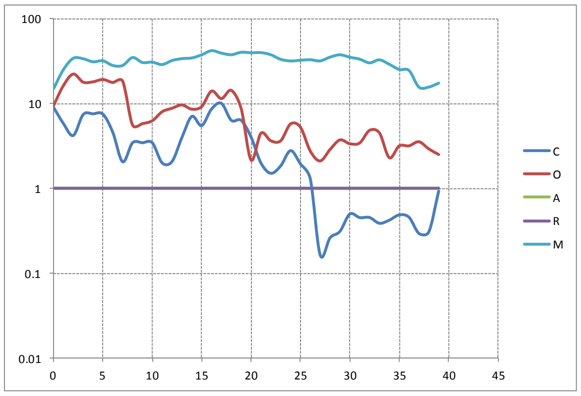

Figure 7: A = R average parameter evolution

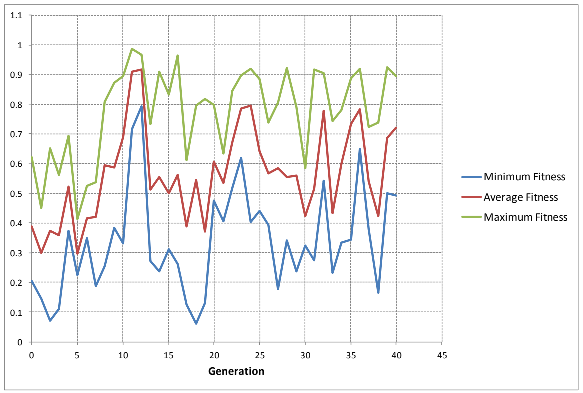

Figure 8: A = R Hall of Fame

It can be seen from these results that the flocking task with A = R is a much more difficult problem to solve than pure flocking. While improvement was shown in the top and average fitness measurements over time, the HoF results imply that very few candidates were found that could give reliably high results. In the case of Figure 8, the best candidates' average fitness value was raised from 0.4 to 0.7, but the spread between the best and worst of the repeated evaluation remained stubbornly at 0.5. Qualitatively similar results were found during other GA runs.

The strategy that was evolved in this case was to maximize momentum, while keeping cohesion and avoidance within an order of magnitude. The C/O ratio could not become too small (~0.2) or too large (~5) so that groups could form. With a high M value acting as a sort of memory for past movements, the small, steady influence toward flocking from the alignment behavior could build up over time as the randomness canceled itself out. Over time a cohesive and aligned flock can be formed despite the presence of randomness.

Exploration

In this run, an 11-member CSO system was allowed to evolve behavior that corresponded to exploration. The mCells were initially placed in a straight line and allowed to move around the field, "discovering" locations on a 100 x 100 grid and turning the discovered locations blue. The fitness function was simply the portion of the field discovered after 200 time steps, expressed as a decimal value from 0 to 1.

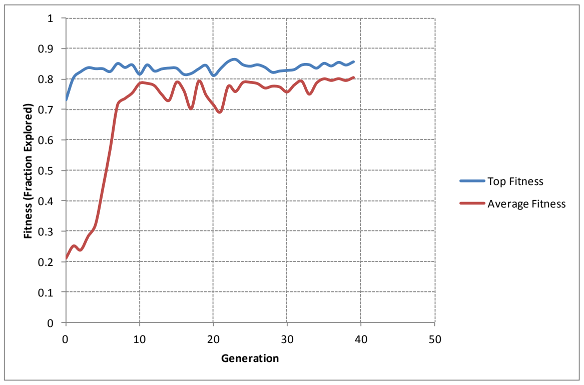

Figure 9: Typical exploration fitness evolution

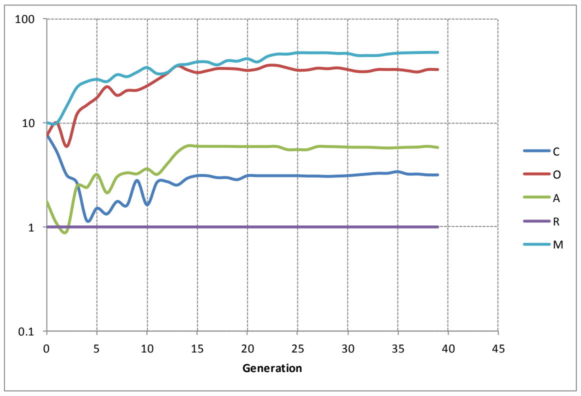

Figure 10: Typical exploration average COARM parameter evolution

It can be seen (Figure 9, Figure 10) that the typical GA for exploration reached a qualitatively steady solution after approximately 20 generations, with most of the dramatic changes occurring within the first 10 generations. This pattern typically held for the exploration problem.

According to Figure 10, the O and M (avoidance and momentum) parameters came to dominate by the end of the GA run. This happened in the majority of flocking GA's, and intuitively fits what a human designer might try if he were asked to assign a set of parameters to accomplish exploration. The high O and M predictably lead to the cells spreading out quickly in all directions and continuing in a set direction until they sense another cell. Then the two colliding cells turn and travel in other directions. This allows cells to work in parallel to discover new territory, and generally stay out of one another's way.

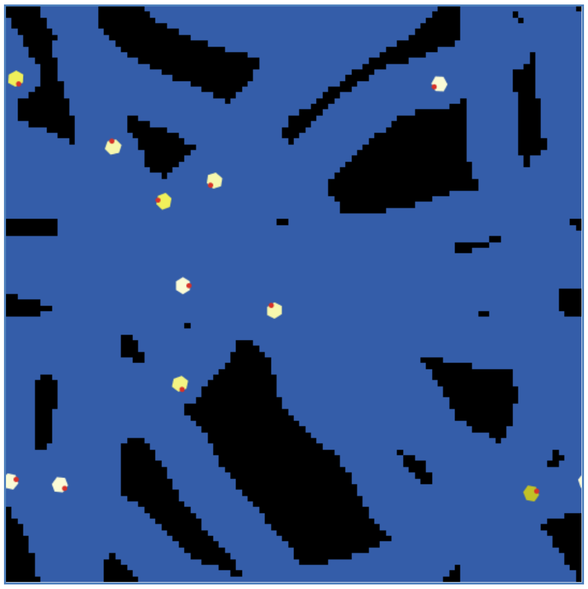

Figure 11: "Intuitive" case with high O and M gives middling results

As is often the case in emergent systems, human intuition fails to create optimal local rules [24]. This intuitive setup performs well (fitness score in the 0.8 to 0.9 range), but improvement can be found using the GA.

At the end of certain GA runs, the dominant parameters were Alignment and Momentum, rather than Avoidance and Momentum. Intuitively, it would seem that such high A values would lead to a single flock which was too cohesive and thus could not send individual cells to explore new territory, but the "clever" output of the algorithm actually resulted in precisely balanced parameters whose C/O ratio, along with high Alignment, caused an emergent fanning and sweeping behavior:

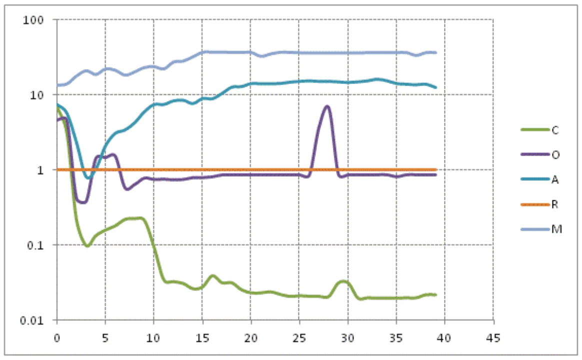

Figure 12: High A and M values with a proper C/O ratio produce fanning and sweeping

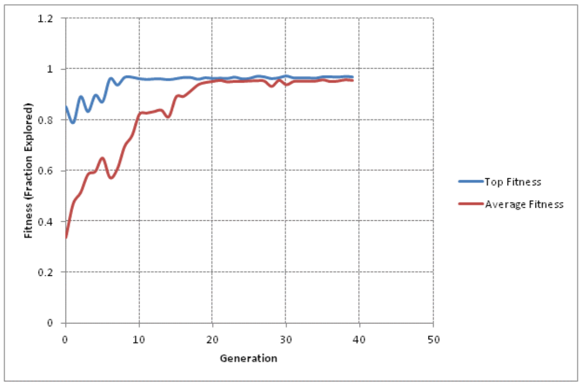

Figure 13: Fitness evolution in fanning GA

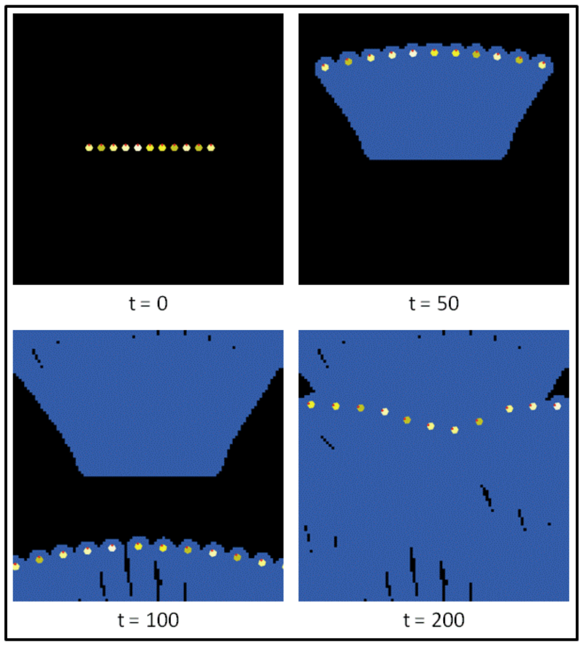

Figure 14: Emergent spreading and sweeping behavior

Compared to the GA's which discovered high O and M behavior, the high A and M behavior discovers approximately an extra 10% of the field. Due to the stochastic nature of the algorithm, it is impossible to determine *a priori* which set of parameters a GA will converge to, if it converges at all. This is why it is important to run multiple GA's on one problem, or insert extra GA steps to ensure a diverse population is maintained. Note also that this sweeping behavior is dependent on the initial conditions of the flock (all in a straight line, facing up), and should not be expected to arise in a population with a different initial configuration.

If we can expect this initial condition in a real-world deployment of exploratory robots, our findings are valid and fan/sweep is a useful emergent strategy for exploration. However, if that assumption is false, then we have fallen for the common trap of optimizing to a specific condition and fitness function, rather than a general design intent [19]. It is quite possible that in other initial configuration, other emergent problem-solving patterns would be evolved.

Percentage Exploration

Encouraged by the results of the exploration GA, we attempted to further challenge the algorithm by selecting for parameter sets that would discover only a certain portion of the field within the allotted amount of time (200 simulation time steps). This forces the system to exhibit some sort of restraint or throttling, as the previous results showed that full-on exploration can reliably discover at least 85% of the field. The target exploration values range from 0% to 75%.

The search for no exploration (0%) predictably led to high-C behavior which prohibited any flockers from leaving the initial pack. This clustering behavior is sometimes discussed in the literature as a desirable behavior in self-organizing systems [31], but here it evolves when the system requirements are to not explore the system's surroundings.

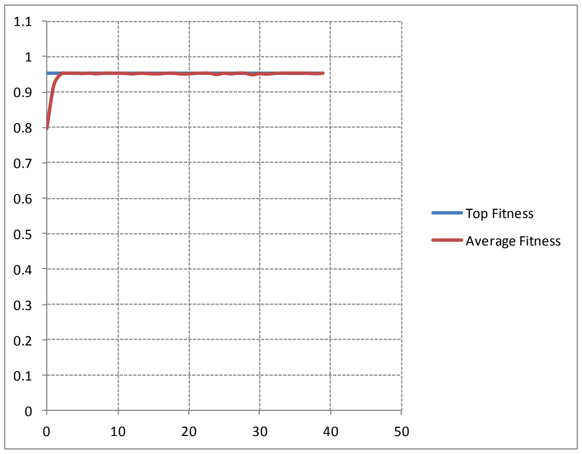

Figure 15: Fitness evolution for 0% exploration

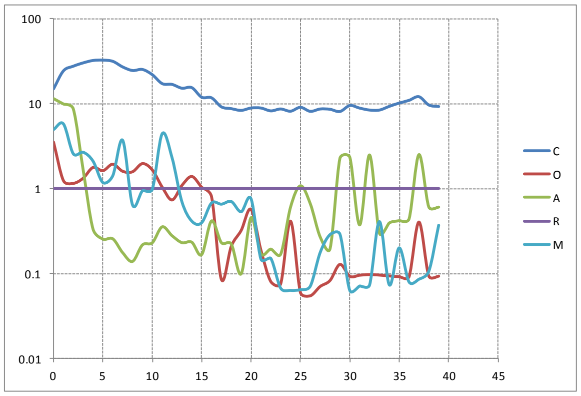

Figure 16: Parameter evolution for 0% exploration

This pattern was repeated for every run of the GA, and there was almost always at least one candidate randomly discovered in the first generation which achieved the maximum possible fitness. These figures are only included to illustrate the point that sometimes a task is actually too simple to justify the computational cost of a GA, and a designer must take these time tradeoffs into account.

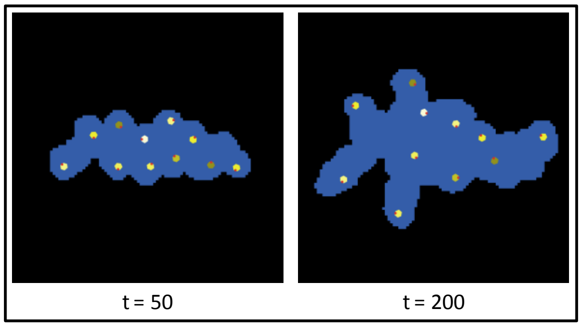

Figure 17: High O strategy for 25% exploration

The search for 25% exploration resulted in more interesting results. These GA's tended to converge to one of two behavioral profiles. In Figure 17, a high-O, low-M behavioral profile was evolved. The strategy for this flock was to immediately disperse so that they were outside of one another's field of vision. Then, once they were "on an island," the random behavior dominated, and they flitted about (R and M are the only behaviors that act when a flocker has no neighbors), uncovering just enough new territory to move the system near 25%.

Another typical evolved behavioral profile is given in Figure 18. Here we see high-C/O, high-A behavior, which causes tightly packed, single-file, finger-like flocks to emerge. The high C/O ratio causes the tight packing, and the high A value ensures that the most stable configurations are long trains. These flocks do not venture far from the initial configuration, but extend out like a pseudopod and explore only enough to reliably uncover 25% of the field's territory within 200 steps.

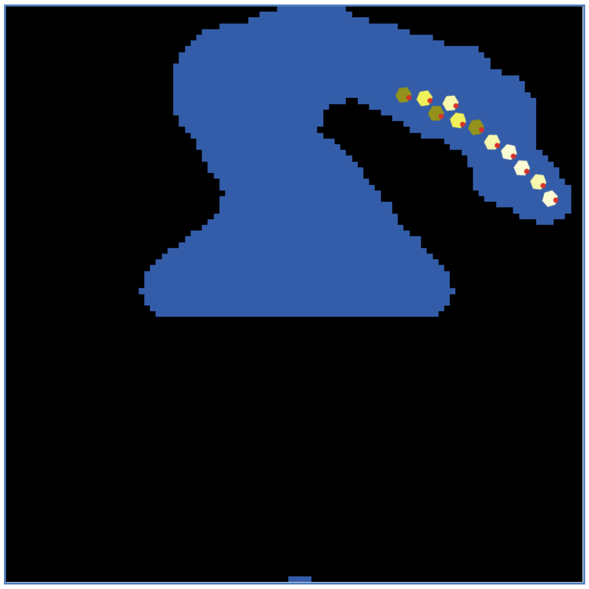

Figure 18: High C/O and high A strategy for 25% exploration

Again we see that a GA can converge to one of two (or more) qualitatively different strategies. The first strategy (high-O, low-M) is dependent on the agents' field of vision, which we fixed for all experiments. If the field of vision were larger, we would expect this strategy to uncover too much of the field, as the agents would pass the 25% mark before their random behavior came to dominate. The second behavior profile (high C/O, high-A) was dependent on the number of time-steps that the system was allowed, as the agents were still active at the 200th step, and would have discovered too much of the field if allowed to run further. These results remind us of the importance of considering fixed variables in the agents' capabilities as well as the translation from simulation to reality in the formation of the fitness function.

The GA that selected for 50% exploration gave results typically resembling the following figures:

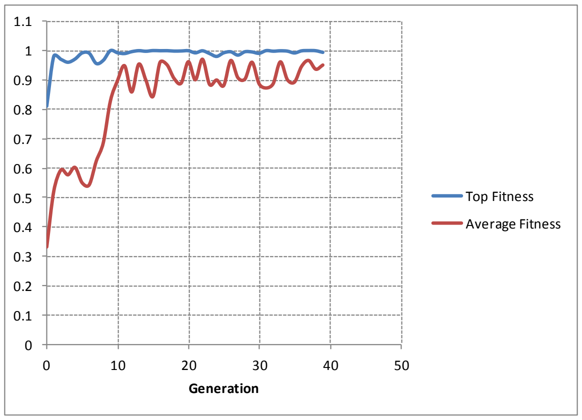

Figure 19: Fitness evolution for 50% exploration

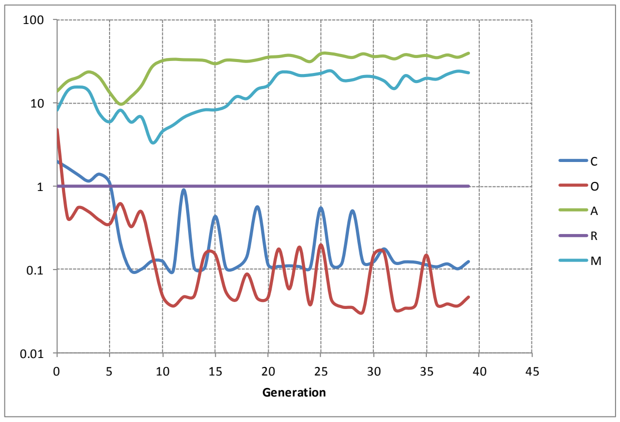

Figure 20: Parameter evolution for 50% exploration

The high-A, high-M behavior allowed a single flock to form and simply travel ahead while maintaining a roughly constant separation distance. This is because in the flock's initial state, the agents' vision covers about half of the horizontal row. If the flockers maintain this formation and simply sweep the field once, they will reliably uncover about 50% of the available field. The end result of this behavior is shown in Figure 21.

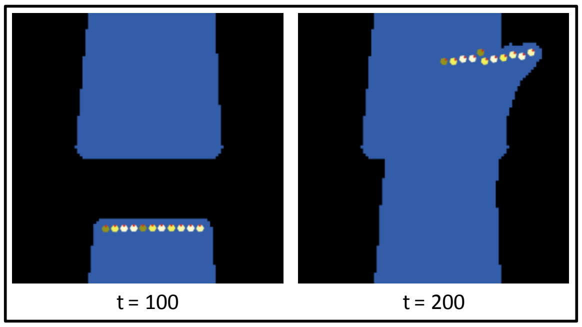

Figure 21: Flock maintenance behavior for 50% exploration

Certain 50% GA's evolved a high-O strategy, similar to Figure 17 for 25% exploration, but with a higher Momentum value so that the flock would spread out more before the random behavior dominated.

The results for 75% exploration are shown in the following figures:

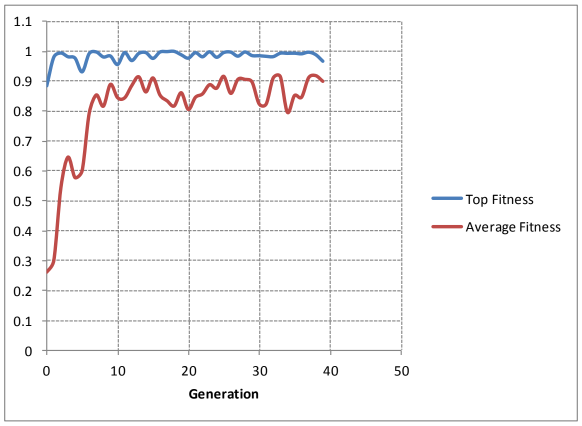

Figure 22: Fitness evolution for 75% exploration

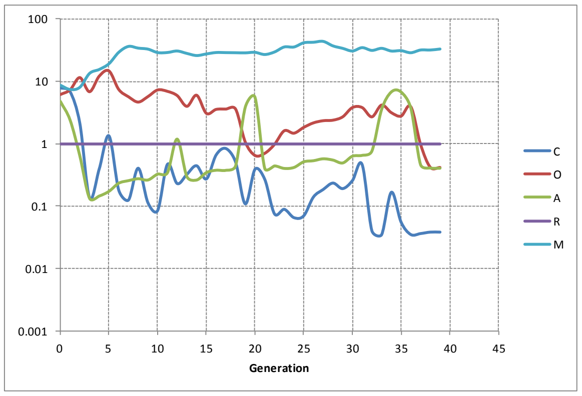

Figure 23: Parameter evolution for 75% exploration

This strategy (high-M, high O/C ratio) was found in all runs of the GA for this amount of exploration. It results in system behavior of spreading out quickly, and then traveling in straight lines while avoiding neighbors, similar to the behavior evolved in Figure 10 for 100% exploration.

CONCLUSIONS AND FUTURE WORK

We have shown that a simple genetic algorithm is capable of tuning the parameters of a CSO system so that its emergent behavior corresponds to the intent of the designer. For simple tasks, such as unrestricted flocking or clustering, hand-tuned parameters performed well, and the use of GA was not shown to be justified, but for more complicated tasks, such as flocking with limited sensory information, or exploration within a set amount of time, a GA produced results that were superior to what human intuition would have suggested.

Out of a search space that contained $2^{32} \approx 4.3 \times 10^9$ possible parameter combinations, no more than 600 simulations were carried out during any run of the GA, and in every run, candidates of high fitness were discovered.

We wish to find more difficult tasks to challenge our CSO further. It will be simple to apply the GA developed here to other simulations in the NetLogo environment. Given a set of parameters, a numerical encoding, and a global fitness function, this approach can be applied to any NetLogo simulation.

The genetic algorithm itself could be made more sophisticated using techniques demonstrated in the literature in the last few decades. Like any optimization algorithm, GA is a tool to be hacked according to the user's unique situation. Pure adherence to the GA's biological inspiration makes for interesting GA research, but many techniques, some even included in the seminal texts on GA [32], [17], are available to the user of GA to improve the algorithm and ameliorate some potential downfalls of the stochastic optimization process. The GA could also work at a higher level on the self-organizing rules. Currently, the 5 behaviors of the mCells are defined, and the GA only optimizes the relative weights of those behaviors. It may be possible via evolutionary optimization to evolve the local rules themselves, in addition to their relative importance.

For some tasks, different runs of the GA would produce qualitatively different emergent behavior in the final generations. For example, for the exploration task, some GA runs would converge to high avoidance behavior sending agents in divergent particle-like trajectories, while other runs would converge to the high Alignment and high Momentum sweeping behavior. It would be useful to develop a means of identifying these different ways of achieving the same goal (both emergent behaviors had high fitness) so that they can be preserved for later study and one or more are not lost in the churn of the GA as its population reaches homogeneity.

Of course the final goal is to create physical CSO systems whose emergent tasks are useful for the general consumer. Work is underway in our laboratory to embed small iRobots with the self-organizing rules described in this paper to test their emergent behavior when limited by actual hardware constraints.

REFERENCES

[1] G. Pahl, K. Wallace, and L. Blessing, *Engineering design a systematic approach*. London: Springer, 2007.

[2] N. P. Suh, *Axiomatic design : advances and applications*. New York: Oxford University Press, 2001.

[3] P. H. Welch, K. Wallnau, A. T. Sampson, and M. Klein, "To boldly go: an occam-π mission to engineer emergence," *Nat Comput*, vol. 11, no. 3, pp. 449–474, Sep. 2012.

[4] Y. Bar-Yam, *Dynamics of complex systems*. Reading, Mass.: Addison-Wesley, 1997.

[5] K. Kelly, *Out of control : the new biology of machines, social systems and the economic world*. Reading, Mass.: Addison-Wesley, 1994.

[6] A. S. Mikhailov, "From Swarms to Societies: Origins of Social Organization," in *Principles of Evolution*, H. Meyer-Ortmanns and S. Thurner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 367–380.

[7] G Zouein, C. Chen, and Y. Jin, "Create Adaptive Systems through 'DNA' Guided Cellular Formation," in *Design Creativity 2010*, 2010, p. 149.

[8] C. Chen and Y. Jin, "A Behavior Based Approach to Cellular Self-Organizing Systems Design," in *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Washington, DC, 2011.

[9] W. Chiang and Y. Jin, "Toward a Meta-Model of Behavioral Interaction for Designing Complex Adaptive Systems," 2011, pp. 1077–1088.

[10] W. Chiang and Y. Jin, "Design of Cellular Self-Organizing Systems," presented at the IDETC/CIE 2012, Chicago, Illinois, 2012.

[11] B. Calvez and G. Hutzler, "Ant Colony Systems and the Calibration of Multi-Agent Simulations: a New Approach," in *Multi-Agents for modelling Complex Systems (MA4CS'07) Satellite Workshop of the European Conference on Complex Systems 2007 (ECCS'07)*, Allemagne, 2007, p. 16.

[12] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," in *ACM SIGGRAPH '87 Conference Proceedings*, 1987, vol. 25–34.

[13] R. Beckers, O. E. Holl, J. L. Deneubourg, Z. Bielefeld, and D.- Bielefeld, "From local actions to global tasks: Stigmergy and collective robotics," 1994, pp. 181–189.

[14] Y. Song, J.-H. Kim, and D. Shell, "Self-organized Clustering of Square Objects by Multiple Robots," in *Swarm Intelligence*, vol. 7461, M. Dorigo, M. Birattari, C. Blum, A. Christensen, A. Engelbrecht, R. Groß, and T. Stützle, Eds. Springer Berlin / Heidelberg, 2012, pp. 308–315.

[15] J. Werfel and R. Nagpal, "Extended stigmergy in collective construction," *Intelligent Systems, IEEE*, vol. 21, no. 2, pp. 20–28, 2006.

[16] W. Chiang, "A Meta-interaction Model for Designing Cellular Self-Organizing Systems," University of Southern California, Los Angeles, CA, 2012.

[17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, 1989.

[18] K. Tumer and N. Khani, "Learning from actions not taken in multiagent systems," *Advances in Complex Systems*, vol. 12, no. 04n05, pp. 455–473, 2009.

[19] B. Calvez and G. Hutzler, "Automatic Tuning of Agent-Based Models Using Genetic Algorithms," in *Multi-Agent-Based Simulation VI*, vol. 3891, J. Sichman and L. Antunes, Eds. Springer Berlin / Heidelberg, 2006, pp. 41–57.

[20] F. Stonedahl and U. Wilensky, "Finding Forms of Flocking: Evolutionary Search in ABM Parameter-Spaces," in *Proceedings of the MABS workshop at the Ninth International Conference on Autonomous Agents and Multi-Agent Systems*, 2010.

[21] B. J. Cole, "Evolution of Self-Organized Systems," *Biol Bull*, vol. 202, no. 3, pp. 256–261, Jun. 2002.

[22] S. van Berkel, D. Turi, A. Pruteanu, and S. Dulman, "Automatic discovery of algorithms for multi-agent

systems," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, New York, NY, USA, 2012, pp. 337–344.

[23] J. H. Miller, "Active Nonlinear Tests (ANTs) of Complex Simulation Models," *Management Science*, vol. 44, no. 6, 1998.

[24] J. P. Crutchfield, M. Mitchell, and R. Das, "Evolving cellular automata with genetic algorithms: A review of recent work.," presented at the First International Conference on Evolutionary Computation and Its Applications, Moscow, Russia, 1996.

[25] G. Narzisi, V. Mysore, and B. Mishra, "Multi-objective evolutionary optimization of agent-based models: An application to emergency response planning," in *The IASTED International Conference on Computational Intelligence (CI 2006)*, 2006.

[26] U. Wilensky, *NetLogo*. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University, 1998.

[27] U. Wilensky, *NetLogo Flocking Model*. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University, 1998.

[28] F. Sadjadi, "Comparison of fitness scaling functions in genetic algorithms with applications to optical processing," 2004, vol. 5557, pp. 356–364.

[29] J. M. Fitzpatrick and J. Grefenstette, "Genetic Algorithms in Noisy Environments," *Machine Learning*, vol. 3, pp. 101–120, 1988.

[30] F. Stonedahl and S. H. Stonedahl, "Heuristics for sampling repetitions in noisy landscapes with fitness caching," 2010, p. 273.

[31] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in swarm-bots," *Robotics, IEEE Transactions on*, vol. 22, no. 6, pp. 1115–1130, 2006.

[32] J. H. Holland, *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, Mass.: MIT Press, 1992.