Proceedings of the ASME 2016 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2016
August 21-24, 2016, Charlotte, North Carolina, USA

IDETC2016-60053

ADAPTABILITY TRADEOFFS IN THE DESIGN OF SELF-ORGANIZING SYSTEMS

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

Newsha Khani
IMPACT Laboratory
Dept. of Aerospace & Mechanical Engr.
University of Southern California
Los Angeles, California 90089
niusha.khani@gmail.com

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

ABSTRACT

Self-organizing systems have great potential for adaptability, but as complex systems, they can also be prone to unpredictable behavior, cascading failures, and sensitivity to perturbations. Also, designing systems for adaptability may introduce overhead that reduces their performance. This paper investigates the design tradeoffs between adaptability and performance in the context of a box-pushing task. Using a genetic algorithm to optimize a parametric behavioral model, we are able to test systems optimized under different conditions for performance and reliability. It was found that a system optimized in the face of random initial conditions and internal perturbations was more robust than a system optimized without these perturbations, showing a higher overall fitness over diverse trials, but it could not take advantage of particular initial conditions that would have allowed it to achieve a one-off high fitness in a repeatable environment. A system optimized with predesigned initial conditions was found to achieve a very high fitness, but when it was retested with random initial conditions, its performance plummeted, indicating that it was fit to the ideal initial conditions but not robust.

Keywords: Self-organizing systems, Genetic algorithm, Optimization, Field-based regulation.

1. INTRODUCTION

Engineered self-organizing systems are composed of elementary autonomous components that can collectively achieve a goal with no system-level knowledge or central control. An example of a self-organizing system would be a swarm of autonomous robots. Typically, the individual robots will be small in size and have only simple capabilities, and a large number of identical robots will be included in the swarm. Self-organization of such systems is becoming an important tool in engineering design, due to a market "pull," and a technology "push."

The market motivation for self-organizing systems is to create adaptable systems. The ability to cope with change – whether in functional requirements, environmental conditions, or the system itself – is now often ordered as a system requirement [1], [2]. This is especially true of systems that will face extreme environmental stressors, such as military applications, systems with lifespans so long that they need to perform new functions after deployment, or systems operating at scales so small that they are difficult to control or repair [3], [4].

Self-organizing systems have the potential to display adaptability through redundancy, scalability, and impermanent connections. Because the components of the system are autonomous, there are no long causal chains of system functionality that can be broken by the malfunction of a single part. If a self-organizing agent breaks down or is removed, another identical agent can take its place. Because the systems are based on local interactions, there is no shared resource (e.g. energy storage or communication bandwidth) that acts as a hard limit to the size of the system. Agents can form local teams of appropriate sizes, independent of the entire system's size, to adaptably scale the system. The agents' ability to form and disband interactions also allows the system to re-organize to better meet changing conditions.

The technology push stems from miniaturization, ubiquitous computing, bio-inspired robotics, and the increasing availability of off-the-shelf robots with interactive capabilities. Enabling technologies like KiloBots [5] and the open-source "Jasmine" robots [6] provide simple robotic agents that can form swarms. Unmanned vehicles for the air and sea are also now commercial off-the-shelf products, and these can work together like flocks

and schools. Another possible substrate for self-organizing systems are hybrid biological-mechanical systems that take advantage of natural organisms' size and locomotion but allow for algorithmic control. For example, researchers have demonstrated digital control over live cockroaches [7] and miniature swimming robots made with jellyfish membranes [8].

These relatively recent technologies can be incorporated into a designer's toolbox if he/she has a methodology for leveraging the interactions among the simple agents to create adaptable systems.

One major difficulty in the design of self-organizing systems is their complexity. Complexity is a result of the interactions of many components [9]. Complex systems can have great functionality, but they are also prone to cascading failures, runaway conditions, nonlinear responses to control inputs, and sensitivity to perturbations [10]. Generally, engineers try to minimize the interactions among the components that they design to avoid the problems that come with complexity [11].

Despite this, self-organizing design allows complexity by giving agents the ability to interact with one another and freely form relationships. It is then designer's duty to bring out the good qualities of self-organizing systems while mitigating the dangers of complex systems. In this paper, we focus on strategies for coping with the sensitivity of self-organizing systems to initial conditions and perturbations. Using agent-based simulation for analysis and genetic optimization, we generate optimized behavior models for self-organizing agents. By tuning the optimization algorithm in different ways (e.g. to search for robust systems), we can investigate the performance costs that come with optimizing for adaptability.

In this paper, we explore a self-organizing system whose purpose is to push a box toward a goal. It is an expansion of our previous work given in [12], [13]. In those papers, the agents were given the ability to broadcast the conditions in their local vicinity, so that the agents could coordinate to move the box. The agents would individually find their chosen locations for the box and move it from there. Then, when all agents had taken a place, they would push it toward the desired locations. The process would repeat until the system was successful or the time limit ran out. At a system level, it looked like a synchronized alternation between formation and pushing.

The expansion in this paper is to complete this task without the aid of system-wide synchronization. Note that synchronization *is* possible to achieve in self-organizing systems (e.g. [14], [15]) but is not guaranteed and requires dedicated hardware and/or algorithms. Because one objective in designing self-organizing systems is to make the hardware as simple as possible [16], good performance without the synchronization overhead represents a notable improvement.

In exploring this problem, we identified an interesting sensitivity to initial conditions and internal perturbations. This is an inherent property of complex systems, and thus a risk in the design of self-organizing systems, but it was effectively covered up by the system synchronization in the previous work. With a new distributed behavior algorithm, we were able to counteract these problems in the unsynchronized system as well. Using a genetic algorithm for optimization, we were also able to determine the performance penalty for systems that were designed for robustness vs. systems that were optimized for particular initial conditions. Using this approach will allow designers to make informed tradeoffs between robustness and performance according to their systems' requirements and environmental constraints.

2. RELATED WORK

There are several examples of coordinated object-moving from the literature and a vast amount of work in the general field of self-organizing systems. In this section, we will present some relevant examples from these areas and summarize our previous work on cellular self-organizing (CSO) systems.

2.1 Natural Self-Organizing Systems

Some of the most impressive self-organizing systems are found in nature. Here, simple organisms like insects, or larger animals following simple rules like birds can display complex group behavior. Flocking in birds is well-known, but perhaps the most impressive display of aerial coordination is found in locusts, whose swarms can reach a size of 1 billion insects [17]. Ants are able to find food, recruit fellow workers to harvest the food, and concentrate the effort along neat lines, all without a central command. They can even cooperate to lift items many times the bodyweight of an individual ant [18], [19]. "Simple" termites cooperatively build mounds with differentiated and interconnected chambers that are larger than any other free-standing structures in the animal kingdom, second only to human construction [20], [21].

2.2 Formation Control Systems

Many artificial self-organizing systems show nature-inspired flocking behavior based on the Boids algorithm [22]. Useful engineering applications of flocking include studies on urban traffic flow [23] and crowd control during evacuations [24]. Military forces are also particularly interested in this type of analysis and control of large fleets of vehicles [25]. Self-organization of unmanned aerial vehicles (UAVs) is another promising application and research field. Self-organizing algorithms have been applied to the control of UAV swarms for target detection [26]. Evolutionary optimization of UAV control algorithms was used in [27] to enable coordinated navigation and obstacle avoidance.

2.3 Moving, Gathering, and Building Systems

In addition to formation control, self-organizing algorithms have been developed to manipulate objects. Zhang et al [28] used a team of 3 robotic fish to move a block toward a goal with a 72% success rate, with similar results given in [29]. Parra-Gonzalez et al [30] used a wave front algorithm for robot motion planning in the box-pushing problem. Trianni [15] used genetic optimization to specify neural network controllers for self-organizing "s-bots" that had to display synchronization and cooperation. Groups of non-communicating robots have been

used to gather pucks in the work of [31] and [32]. Werfel [33] showed that self-organizing robots can use stores of building materials to build pre-specified shapes.

2.4 Cellular Self-Organizing Systems

In our previous work on the design of adaptable systems, we have focused on simulations of robotic agents with unsophisticated hardware. The functionality is an emergent property of the interactions among primitive agents, just as complex life-forms are built from the interactions among simple cells. Thus we use the term *Cellular Self-organizing Systems* (CSO) to describe this work. The first CSO systems were made to display reconfiguration between two pre-specified shapes [34]. Emergent formations such as flocking and aggregation were found with parameter changes in [35]. Simple box-pushing through obstacles using field-based regulation was shown in [36], followed by a more sophisticated study with communicating agents [13]. Optimization of parametric behavioral models has been used to develop flocking, exploration, foraging, and a protective convoy in [37], [38].

In the studies mentioned in this section, different aspects of adaptability have been demonstrated, but as the systems increase in complexity, inherent properties of complex systems may pose threats to system performance. These tradeoffs have not been studied in detail, and they are the focus of the current paper.

3. THE BOX-PUSHING PROBLEM

It is possible to achieve desirable system functionality from minimalist agents [16]. A motivating case study can be found in nature, where the advantage of coordinated object-moving is evident in the behavior of foraging ants, which have been reported to transport food objects that are heavier than a single worker ant by a factor of 5000 (or heavier than the collective team of ants by a factor of 50) [39]. By analogy, in the engineered world, debris removal, mining, or construction could all be aided by cooperating robots.

The Box-Pushing task belongs to a type of task that is often described as the "piano mover's problem" or path planning, which has been studied in the robotics, artificial intelligence, and control theory communities [40]. While this problem is often approached from a mathematics or topological perspective [41], in the present work, the agents are trying to solve the problem collectively using local information.

Figure 1 shows the simulated environment for the box-pushing task. The simulation is run using NetLogo [42], a popular platform for agent-based simulations.

The brown box is to be pushed to the *goal* (concentric black circles) by the *agents* (green squares). They can push the box to move and rotate it. The box cannot be pushed through the red *obstacle* or the blue *walls*.

The system is limited by the total distance traveled by agents (representing battery power or abstract efficiency metrics) before it turns off. In keeping with our previous work [12], if the centroid of the box reaches the same x-coordinate as the goal, the trial is considered a success.

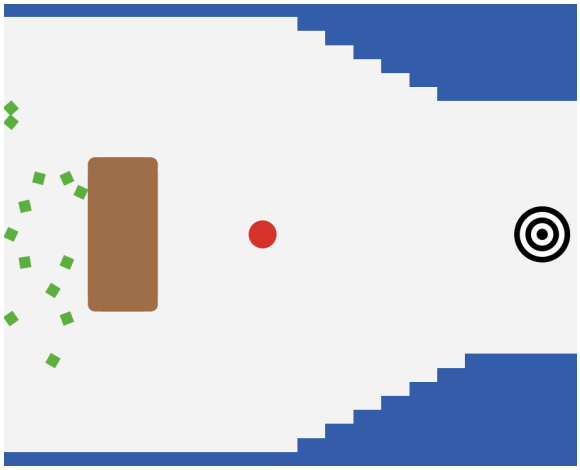

Figure 1: The Box-Pushing Problem

Because the NetLogo simulation software is dimensionless, we will refer to dimensions as being measured in patch-widths (pw) throughout this paper. A patch is an elementary square area of the NetLogo environment. For reference, the agents are 1 pw large along the diagonal; the box is 5 pw wide; and the starting distance from the center of the box to the goal is 30 pw.

The simulation rests on a simplified physical model. At every simulation time-step, the forces from every pushing agent are summed. Every push carries an equal force, and a vector sum of 11 pushes will move the box 1 pw in a given direction. The torque is applied around the centroid of the box, and a push with a moment arm of 5 pw will rotate the box 1 deg. The translation and rotation scale linearly with the number of agents pushing and their moment arms, respectively. If the box is moved into an agent, that agent is pushed out of the way. If the box or an agent is moved into walls or obstacles, its motion is halted, but no other reaction force is simulated. Agents cannot move through the box, walls, or obstacles.

An agent is assumed to be able to move in any direction in the horizontal plane. They can broadcast and receive information using one-to-many signaling, but not direct one-on-one communication. They can differentiate other agents from the environment, and discriminate among environmental stimuli. They can measure distances and directions, and have enough computing power to perform simple reasoning algorithms. They have enough data storage to govern state changes. They have a maximum speed of 3 pw per simulation time-step.

These assumptions are in keeping with our previous work on CSO systems [35], and they are similar to the definition of a "minimalist" robot [16]. They are also reasonable with respect to current swarm robot hardware (e.g. [43], [44]).

3.1 Key Problem Features

With a large box and simple agents, it is necessary for agents to work together to move the box. This tight coordination is difficult to design in a system if it is assumed that agents' communication is limited. How can this coordination be achieved by independent agents? What is the minimal communication required?

Success in the task is partially stochastic. Agent initial conditions and internal perturbations (common in complex and self-organizing systems [45]) will affect the system's behavior. How can the system be made more robust to these perturbations? And does it lose efficiency by seeking adaptability?

3.2 Design Methodology

Several helpful methodologies for the design of self-organizing systems have been proposed [46], [47], but no standard exists yet. We use the design methodology shown in Figure 2. Because the system-level form is outside of the designer's control, and the agent-level hardware is meant to be quite simple, the agent-level behavior is the designer's key point of influence over the system and the focus of our methodology. The methodology is similar to other design or systems engineering methodologies (e.g. [48], [49]) but with several distinctive features tailored to self-organizing systems:

- **Agent behavioral design:** this is the defining feature of self-organizing systems. The system components are not inert. They are programmable, and their behavior, not just their physical form, can be designed [50]. The behavioral capacity is a set of behavioral primitives, and the behavioral selection is an algorithm for applying the behavioral primitives at a given time. We recommend the use of field-based behavioral design for this stage, which will be discussed in more detail in Section 4. The output of this step is a parametric behavioral model that can be simulated and optimized.
- **Simulation/Optimization:** the complex dynamics of self-organizing systems cannot be captured analytically [51]. Therefore simulation is included as a mandatory step in the design process. Because we leave detailed design to this stage, the simulation can be integrated with optimization algorithms for fast optimization of behavioral parameters [37]. By encoding the behavioral design with artificial DNA, we can optimize it with a genetic algorithm.

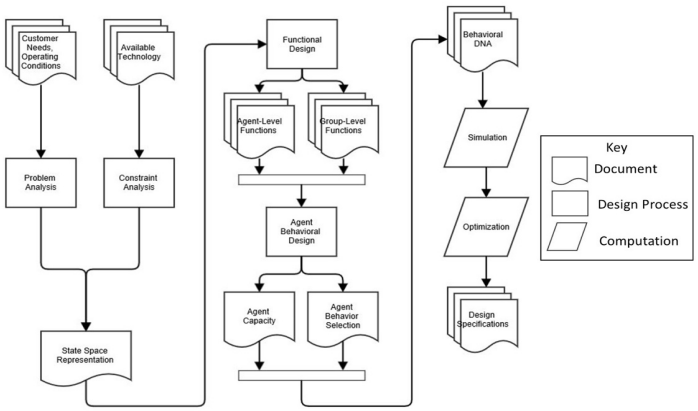

Figure 2: Methodology for the Design of Self-Organizing Systems

3.3 Potential Failure Modes

During exploratory work, design of the system was adapted from previous studies on flocking self-organizing systems [50], [52]–[54]. Agents' reactions to each other and the box were based on attraction and repulsion, which are classical primitives in flocking [22]. The agents also simultaneously considered attraction to the goal, and repulsion from walls and obstacles. These behaviors were parameterized, and the parameters were tuned using a genetic algorithm. We will not give details of this exploratory behavioral algorithm here, but a visual summary with motion trace examples is given in Figure 3.

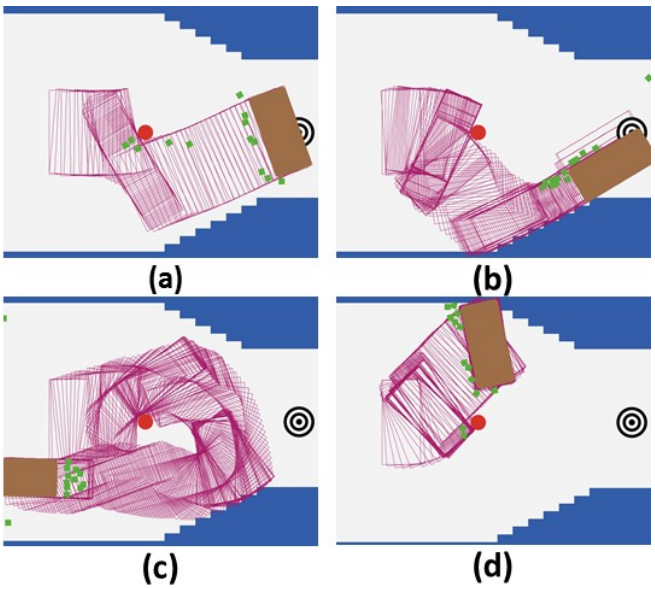

Figure 3: (a) task completion with obstacle collision (b) task completion using walls as guide (c) task failure, pushing box backwards (d) task failure, jamming box against wall

It can be seen in Figure 3 that the system occasionally behaved in critically counterproductive ways, pushing the box backwards or jamming it against a wall. Another behavior, not shown, was agents' failure due to abandoning the box and moving to the goal without it. Even in successful attempts, the box usually collided with walls or the obstacle.

Note that all the behaviors described here were found in systems *after* optimization. Because the GA could only optimize within the parameter space given to it, it was clear that the conceptual design of the system needed to be reworked to minimize failures. The previous work [12] avoided these failures by synchronization. In that study, agents would choose a location at the perimeter of the box first. Then, when all agents had found a suitable location, they would communicate and decide whether or not to push the box. In the current work, we attempt to create a new behavior algorithm that can avoid these pitfalls even without system-wide synchronization. With this goal and potential failure modes in mind, we present a formal behavioral model for the box-pushing agents in the next section.

4. A SELF-ORGANIZING BOX-PUSHING SYSTEM

No practical design process is as linear as the progression in Figure 2 indicates [55]. Iterative feedback loops also exist due to knowledge generation throughout the process. In the case of the preliminary work mentioned earlier, unsatisfactory results after optimization and new knowledge of the emergent system pitfalls led to rework and a new strategy at the behavioral design phase.

We design the agent behaviors using a field-based approach [36]. A field is a mathematical abstraction of every stimulus that an agent considers. Smooth field functions allow agents to follow local gradients toward goals. A *task field* and *social field* are used. The task field (tField) is a response to objects in an agent's task and environment, and the social field (sField) arises from agent-agent interactions. The fields are treated as two separate concepts, because the sField can be used to dynamically create task-based structures, while the tField controls where these structures should be deployed [38].

The stimuli in the task field are the box, walls, goal, and obstacle. Agents need to consider the desirability of their own location within the field, and the desirability of the box's location within the field.

4.1 Field for Box

The field governing the desirability of the box's location is a pure tField, whose function is given here:

$$field_{\text{box}} = V_{\text{goal}} e^{-\lambda_{\text{goal}} d_{\text{goal}}} + V_{\text{obs}} e^{-\lambda_{\text{obs}} d_{\text{obs}}} + V_{\text{wall}} e^{-\lambda_{\text{wall}} d} \quad (1)$$

where V and λ are behavioral design parameters, and d is the distance to the goal, the center of the obstacle, or the nearest point on the wall. Because the terms in the field equation decay exponentially, the influence of any one stimulus can peak at a point and be negligible elsewhere. This makes it simple to create a local maximum around the goal, and local minima near the walls and obstacle. Thus, with proper selection of decay constants, following an increasing gradient can lead through the environment, between walls and obstacles, and toward the goal. One graphical example applied to the simulation environment is given in Figure 4.

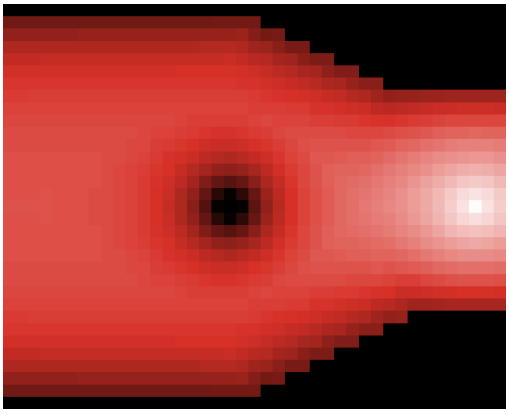

Figure 4: Graphical representation of the field value at every NetLogo patch. White patches have the highest field values. Darker shades of red have lower values, and the black patches represent walls and obstacles.

4.2 Field for Agents

Agents cannot simply follow the field described in equation (1), as they must be mindful of the box's location and orientation within the field as well. The field governing agent movement should distribute them around the box, so that they can protect it from collisions and collectively push it toward higher field values.

The agents use a combined tField and sField to calculate their own movements:

$$field_{\text{agent}} = w_{\text{p}} \sum_{i \in \eta} p_i + w_{\text{b}} b$$

$$p_i = \begin{cases} \frac{d_{\text{p}}^i}{d_{\text{pn}}} & d_p^i \le d_{\text{pn}} \\ \frac{d_{\text{pn}}}{d_{\text{p}}^i} & d_p^i \ge d_{\text{pn}} \end{cases} \quad (2)$$

$$b = \begin{cases} \frac{d_{\text{b}}}{d_{\text{bn}}} & d_b \le d_{\text{bn}} \\ \frac{d_{\text{bn}}}{d_{\text{b}}} & d_b \ge d_{\text{bn}} \end{cases}$$

where η is the set of the calculating agent's neighbors; d_p and d_b are the distance (pw) to an agent or the box, respectively; d_{bn} and d_{pn} are design parameters representing the nominal distance that an agent attempts to maintain from the box and other agents, respectively; and w_p and w_b are relative weights that an agent places on maintaining a distance toward its neighbors, or the box, respectively, if both cannot simultaneously be the nominal distance.

As shown in Figure 5, the box is divided into 6 zones to focus the agents' attention on important interactions and to minimize interference within the system [13]. For an agent to determine its neighborhood η, it seeks agents in its own zone, and if it is on a long side of the box, neighbors in the adjacent zone. For example, in Figure 5Figure 5, where the red agent is calculating its move, η would contain all agents on the right side of the box.

Similar to [37], the equilibrium distances d_{pn} and d_{bn} are used so that agents remain near the box, but also spread out from one another. The intention is for agents to maintain a formation surrounding the box.

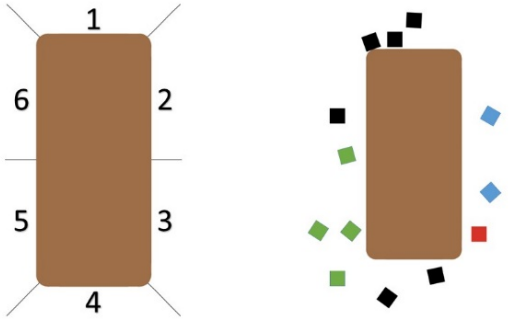

Figure 5: LEFT: the box is divided into 6 zones. The zones determine which agents work together. RIGHT: The red agent is calculating its move. Since it is in zone 3, it considers information from the blue agents, which are in zones 2 and 3; and the green agents, which are in zone 5

4.3 State Changes

The green agents in Figure 5 are opposite the red agent and govern the red agent's state change. State changes separate in time the agent's formation keeping from box pushing. The agent's logic for switching between the two states is simple: at any timestep, the agent will receive the box field values from its neighbors on the opposite side of the box (cross_neighbors in the pseudocode). If the average of these values is higher than the box field at its own location, it will set its state to *pushing*. Otherwise, it will set its state to *forming*. If there are no cross_neighbors, the agent will randomly switch states, according to the probability *switch_prob*. The following block of pseudo-code describes this behavioral algorithm:

```
1: if any cross_neighbors then
2:    signal ← average(box_field of cross_neighbors)
3:    if signal > box_field then
4:       state ← PUSHING
5:    else
         state ← FORMING
6:    end if
7: else
8:    if random(100) < switch_prob then
9:       switch state
10:   end if
11: end if
```

4.4 Summary of Agent Behavior

At a given simulation timestep, an agent will first sense its tField, broadcast its tField value, determine its neighbors on its own side of the box, and determine its cross_neighbors on the opposite side of the box. It will then possibly change its state. If it is in the pushing state, it will move toward the box, and if it reaches the box, it will push. If it is in the forming state, it will move to the point with the highest field value according to Equation (2). Agents can move no further than 3 pw in 1 timestep. All agents perform the same behavioral algorithm in parallel at every timestep.

5. CASE STUDIES

This section describes the goals and methods of this research effort and presents three sets of case studies. There are two related questions motivating this research: how can systems be made more adaptable in the face of perturbations? And what are the tradeoffs in other measures of system performance when designing for adaptability?

5.1 GA Setup

A genetic algorithm (GA) is used to optimize the system parameters. This partially stochastic algorithm was chosen due to the large search space and nonlinearity of the optimization problem. It has been shown to be successful in other similar applications [54], [58], [59].

DNA Encoding

The behavioral parameters are encoded as 8-bit binary numbers. This is the "design DNA" of the system that the GA can optimize. Table 1 shows the values and mapping functions that the GA used for determining behavioral parameters.

Table 1: Mapping GA DNA encoding to behavioral parameters

Parameter	Raw Range	Mapping Function	Parameter Range
V_{goal}	0–255	$f(x) = 1.0369^{127-x}$	0.01 to 100
V_{obs}, V_{wall}	0–255	$f(x) = -1.0369^{127-x}$	−0.01 to − 100
λ_{goal}, λ_{obs}, λ_{wall}	0–255	$f(x) = 3 \cdot \frac{x}{255}$	0 to 3
w_{p}, w_{b}	0–255	$f(x) = 1.0369^{127-x}$	0.01 to 100
d_{pn}, d_{bn}	0–255	$f(x) = 1 + 49 \cdot \frac{x}{255}$	1 to 50
switch_prob	0–255	$f(x) = 100 \cdot \frac{x}{255}$	0 to 100

The several exponential mappings were used for parameters that encode a relative weight. With an exponential mapping, parameters have a median value of 1, and can increase or decrease by a factor of 100.

Genetic Operators

The GA used in this research can be described as a Simple Genetic Algorithm (SGA) [56] with fitness scaling, uniform crossover, and elitism. An SGA randomly generates a population of candidate solutions (the binary strings encoding behavioral parameters) and evaluates them according to a fitness function (measure of simulated global performance). With fitness scaling, the raw fitness scores are scaled so that the best candidate has a score that is a predetermined factor (15, in this study) of the average score, and all other candidates' fitness scores are scaled linearly. This completes the creation of one GA generation.

Using elitism, the top candidate from a generation is cloned (copied bit-for-bit) to the next generation. To fill the rest of the next generation, candidates are randomly selected to mate with one another. The selection probability is proportional to their scaled fitness. The offspring are created by mating two candidates. Under uniform crossover (as defined in [57]), a bit is randomly chosen from either parent to write each bit of the offspring genome. For all non-clones, every bit also has a possibility of mutation (flipping a bit from 1 to 0 or *vice versa*) according to a predetermined percentage (1%, in this study).

The process is continued for a number of generations, until highly fit candidates are found. These operators of selection, crossover, and mutation are meant to generate better and better candidates in progressive generations.

Fitness Function

The pass/fail nature of the task makes it difficult to derive figures of merit. In its most basic form, the task is simply to push the box toward a goal, with no stipulation on how this is to be done. A wide variety of strategies can be used successfully, and the success of any one attempt relies partially on random chance. At the end, the result is either a pass or fail, sorting all systems into one of only two categories. To aid in differentiation of systems, more criteria were added to the system performance metrics: effort, collisions, and reliability.

Effort is the total distance travelled by the agents in the system. Systems can reduce their effort by, e.g., taking the shortest possible route between the origin and the goal, and not backtracking. It is included in the fitness function as a simulation stopping condition. After the system has expended a pre-

determined amount of energy, the simulation is stopped and the fitness is calculated.

Collisions is the number of times that the box collided with the walls or obstacles. It is included in the fitness function as a penalty that reduces the score of a success.

Reliability is tested by allowing the system to complete the task multiple times before the effort budget is exhausted. After a success, the task and environment will reset, allowing another opportunity for the system to complete the task, obtaining a higher fitness with every subsequent success.

For clarity, in the remainder of this paper, any single attempt at pushing the box to the goal will be referred to as a *sprint*. The set of sprints that occur before the effort budget is expired will be referred to as a *trial*. Systems can add to their fitness score with each successful sprint, but the GA only considers the final fitness score at the end of a trial.

A trial is ended after the agents have collectively travel 40000 pw. The fitness is calculated according to Equation (3):

$$fitness = 100 \sum_{i=1}^{M} 0.75^{x_i} \quad (3)$$

where M is the number of successful sprints in a trial, and x_i is the number of collisions that occur during sprint i. It can be seen that the collision penalty is 25%.

Finding Optimal Systems

Because the behavior of the system is partially stochastic, determining the optimal system is not as simple as choosing the system with the highest fitness value found by the GA. The mean and variance of the system behavior must also be considered. Also, different GAs with the same fitness function and search space can converge to different parameter sets, making optimality hard to define. In this work, multiple GAs are run to identify highly fit candidate systems. These systems are then re-tested for reliability, and the system with the highest 30th percentile score (out of 100 trials) is chosen as the optimal candidate. The finalists are determined as follows:

1. 3 GAs are run with random starting populations
   - Choose as finalists
     i. Best candidate from final generation
     ii. 2nd best candidate from final generation
     iii. Best candidate overall
2. A "hall of fame" GA is run with the finalist candidates from the first 3 GA runs used as seeds in the 1st generation
   - Choose 3 finalists similarly
3. All 12 finalists from the first 4 GAs are retested for 100 trials
   - Choose as optimal the candidate with highest 30th percentile fitness

This means that any fitness score reported for an optimized parameter set can be interpreted as having 70% reliability.

5.2 Baseline: Random Initial Conditions and Stepping Order

The baseline setup for the system is for the agents to assume a random initial position on the left side of the box, and for the simulation to update the agent positions in a random order.

This scenario was chosen to include several sources of randomness, and to compare to the previous work in [38]. Several questions guided this case study: Can an unsynchronized field-based system be as reliable as a logic-based system with synchronization? What are the performance advantages and disadvantages of each approach?

Optimized System

Table (2) shows the optimized parameter set found in the baseline scenario.

Table 2: Optimized parameter set found in the baseline scenario

Parameter	Value	Parameter	Value
V_{goal}	5.107	w_{p}	29.43
V_{obs}	-28.04	w_{b}	16.55
V_{wall}	-0.8970	d_{pn}	40.01
λ_{goal}	0.5059	d_{bn}	2.729
λ_{obs}	2.553	$switch\_prob$	100.0
λ_{wall}	2.6706		

In 100 trials, this system had a 30th percentile fitness of 1565.9. Its minimum, mean, and maximum fitness scores were 0, 1572.7, and 1901.3, respectively.

Comparison to Previous Work

Compared with the previous work [13], the unsynchronized system was found to be quite competitive, and more efficient from an energy viewpoint. It was, however, more prone to collisions. Figure 6 shows three trends as a function of the system's energy budget: the raw success rate of the current work, the success rate if sprints with collisions are not counted, and the success rate of the previous work (where collisions never occurred in 100 trials). The x-axis of "Agent Travel Limit" indicates how much energy (measure in patch-width (pw)) is allowed for the agents to use during each simulation. When the allowed energy is low, the agents can succeed only a few times out of the total number of trials.

It can be seen from Figure 6 that the unsynchronized system could complete the task much more reliably with a small energy budget, but it was prone to collisions. At higher energy levels, the synchronized system had more collision-free successes. This is because the synchronized system expended a lot of energy to create formations around the box, and because it showed a lot of backtracking and oscillations in box motion. In the synchronized system, agents could move to an entirely new location around the box during every pushing sequence. In the unsynchronized system, when agents found a location around the box, they tended to stay in the vicinity, minimizing formation-keeping movement. Also, there were fewer oscillations and less backtracking, so the box was pushed to the goal more quickly.

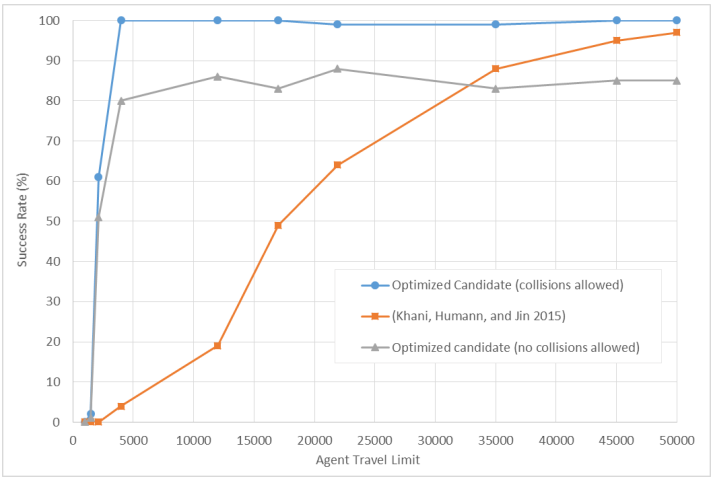

Figure 6: Success rate as a function of energy budget, comparing the optimized, unsynchronized system with the synchronized system

RNG Seed Attached to Candidates

Our next optimization removed the randomness from the simulation by allowing candidate solutions to fix their random number generator (RNG) seed. To do this, the parameter sets were appended with 8 more bits to encode the RNG seed, and this seed was passed to NetLogo at the beginning of a trial.

This seed controls the three sources of randomness within the simulation:

1. Random initial positions
2. Random stepping order of agents
3. Random switching between states if *switch_prob* is not 0 or 100%

The random stepping order is an artifact created by the simulation software. As a serial-processing machine, the computer can only emulate the parallel actions of SO systems by updating the system state in small intervals, and agent states must be updated one at a time to check for collisions and interference. Nonetheless, this simulation artifact does emulate some of the internal perturbations of distributed hardware systems such as unequal agent speeds, missed communication, interference, and unsynchronized update times.

By encoding the RNG seed with the parameters, initial positions and stepping orders can be maintained in clones and offspring of highly fit candidates. They are still generated based on the RNG seed (i.e. not specifically designed), but for any given seed they are repeatable.

Intuitively, this could lead to one of two scenarios:

1. An RNG seed is found that enables advantageous initial positions and stepping orders for a large swath of candidates.
2. Or a coevolved pairing of the RNG seed with parameter values is found that gives high fitness, even though individually the parameter values or the RNG seed may not be generally helpful.

Note: NetLogo uses the Mersenne Twister RNG, as developed by Matsumoto and Nishimura [60], and implemented in Java code by Luke [61].

The random initial positions represent a designer's uncertainty about the system state at deployment. Many self-organizing systems are envisioned to be deployed in environments were precise control is impossible [4], [62], [63]. Then the questions can be: How can a GA optimize in the face of random initial conditions and internal perturbations? What is the performance penalty caused by the use of random initial positions?

These questions have implications in design tradeoffs and optimization approaches. If the designer has to pay a high premium for precise control over the system's initial condition, then it may be wise to forgo that precision if the system is robust to changing initial conditions.

Optimized System

This system evolved to a small equilibrium distance with the box of 2.53 pw. A distance of 1 pw is necessary to avoid colliding with the box. It was generally quick to surround the box, and pushed it toward the goal with minimal stalling or collisions. With its evolved RNG seed of 210000, it was able to achieve a fitness of 2228.4 with 25 complete sprints.

The repeatable initial conditions resulted in a fitness score higher than any found in the tests of the system with random initial conditions. The baseline optimized candidate had a maximum fitness of 1901.3. The candidate with attached RNG seed had a maximum fitness of 2228.4. This indicates that there is at least a 17.2% performance penalty due to random initial conditions and perturbations, if the designer is only concerned with a best-case scenario (a *maximax* optimization).

Table 3: Optimized parameter set for RNG seed added scenario

Parameter	Value	Parameter	Value
V_{goal}	0.01926	w_p	0.02371
V_{obs}	-0.1576	w_b	0.01286
V_{wall}	-0.01118	d_{pn}	23.48
λ_{goal}	0.2471	d_{bn}	2.537
λ_{obs}	1.635	*switch_prob*	90.98
λ_{wall}	2.212	RNG seed	210000

Retesting without Optimized RNG Seed

This performance mentioned above was highly dependent on the RNG seed, however. To investigate this, a retest was performed without the evolved RNG seed (a random new seed was generated for each trial). This parameter set was prone to the same errors as other systems, occasionally jamming the box against a wall or pushing it backwards away from the goal, indicating that it was not robust to changing initial conditions. The particular evolved RNG seed had just prevented the system from displaying this error within the time limit. The lack of robustness gave it a mean performance of 1312.5. Compared to the baseline mean, 1572.3, there was a 16.5% performance penalty due to lack of robustness to an uncertain environment.

Ideal Initial Conditions

Because successful systems tended to surround the box and signal when it was approaching a wall or obstacle, the task completion could be aided by intentional arrangement of agents in positions surrounding the box. One advantageous initial formation has agents on all four sides, regularly spaced, with more agents on the left than the right (because they need to push toward the right). This designed scenario is the focus of the final case study.

The obvious advantage here is that the designer can set initial conditions that are conducive to system functionality, rather than making the system assemble itself. The possible downsides are again related to cost and robustness. This precise control may come at a high price if the system is deployed in remote or harsh environments. Also, if the system behaviors are designed for ideal initial conditions, they may fail in off-nominal cases. As is often the case in design, a cost/benefit decision must be made. These research questions aim to explore the cost/benefit tradeoffs:

- What performance gains are achievable by prescribed initial conditions?
- How severe is the loss in performance when a system optimized for ideal initial conditions is deployed in an off-nominal configuration?

Optimized System

The best candidate found had a fitness of 6500, much higher than any other systems mentioned in this paper. It achieved this by successfully completing 65 sprints without colliding with any walls or obstacles. The candidates evolved in this scenario not only had high fitness, but also were remarkably reliable. Figure 7 shows the initial formation, and a wire trace of 5 sprints from this candidate. Table 4 shows the optimized parameter values.

Table 4: Optimized parameter set for ideal initial coniditons scenario

Parameter	Value	Parameter	Value
V_{goal}	16.88	w_p	17.78
V_{obs}	-0.02975	w_b	74.99
V_{wall}	-2.2193	d_{pn}	39.62
λ_{goal}	0.2471	d_{bn}	1.000
λ_{obs}	0.8000	$switch\_prob$	86.27
λ_{wall}	1.082		

All 100 subsequent trials of the optimized candidate displayed the same behavior as in Figure 7. There were no collisions with the wall or obstacle. The only notable variation from sprint to sprint was whether the system would go above or below the obstacle.

The system achieved such high fitness by maintaining a very tight formation around the box. Its d_{bn} value (governing the equilibrium distance from the box) was 1.0, the lowest possible value in the parameter range. The formation surrounding the box and the state-switching strategy reliably caused agents to move the box to the goal, and the tight equilibrium distance minimized the distance that agents would travel, allowing them to use their energy budget for more sprints.

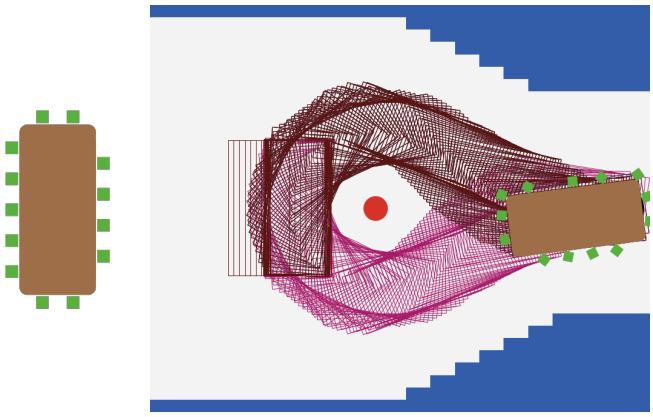

Figure 7: (LEFT) Initial designed formation (RIGHT) Behavior of optimized system. A wire trace is shown of 5 sprints, with a different color for each sprint.

Retesting without Ideal Initial Positions

The optimized parameter set was highly dependent on the initial conditions. In a re-test of the optimized system (N = 100) with the random initial positions of the baseline scenario, the average fitness was 0.004323, with only 54% of trials resulting in any successful sprints. The best trial resulted in a fitness of 0.1784. Agents did a poor job of distributing themselves around the box, often grouping together on only 1 or 2 sides and collectively jamming the box into a wall. Even during successful sprints, the high number of collisions often brought the fitness score down into the thousandths or lower.

6. FINDINGS AND IMPLICATIONS

In this section, we revisit this paper's research questions in light of the data generated from the case studies.

Can a field-based unsynchronized system be as reliable as a logic-based system with synchronization?

The answer is a qualified "yes". In the case where ideal initial positions can be prescribed, the unsynchronized system in this paper is more reliable than the synchronized system of the previous work. Even in the baseline case, the unsynchronized system was very reliable with small energy budgets, and was close to 100% reliable at higher energy budgets.

What are the performance advantages and disadvantages of each approach?

The synchronized system was always free from collisions with the walls and obstacles, but the baseline system was prone to collisions, which could be a major problem, depending on the application. This weakness could be mitigated by better control of initial conditions, or it may be possible to use the same parameter ranges with a new optimization that has a higher fitness penalty for collisions to find systems that are less prone to collisions.

What is the performance loss caused by random initial positions and perturbations? And is there a performance penalty if the system is optimized for robustness to initial conditions?

The RNG-encoded optimization produced a system with high fitness. However, when the same candidate was re-tested without its optimized RNG seed, its maximum fitness out of 100 trials was 14.7% lower, and its mean fitness was 41.1% lower, indicating substantial performance drops due to randomness and perturbations.

The RNG-encoded optimized candidate had a fitness 17.2% higher than any trial of the optimized baseline system, so the performance penalty from optimizing for robustness is that the GA is restricted from seeking unique cases that cause high fitness. This would be interpreted as a performance penalty in a maximax optimization.

From an optimization standpoint, this also serves as a warning to include sufficient randomness in the optimization algorithm if there is uncertainty in the environment, because the optimized parameter set was uniquely suited to the optimized RNG seed, but was not robust to changing conditions.

What performance gains are achievable by prescribed initial conditions?

There is a significant performance gain when the designer is allowed to pre-specify initial conditions. The optimized system with agents surrounding the box in the beginning completed 65 sprints with no collisions in 1 trial, for a fitness of 6500, and it reliably repeated this performance 100 times in further testing. Compare this to the optimized baseline system, which achieved a maximum fitness of 1901.7 in 100 trials, and was prone to collisions in almost 10% of its sprints.

How severe is the loss in performance when a system optimized for ideal initial conditions is deployed in an off-nominal configuration?

The tradeoff to the performance gain from ideal initial conditions is that systems optimized with these conditions have critically degraded performance when deployed in environments with random initial conditions, indicating that they are brittle, or non-robust. As evidence, when re-testing the optimized candidate from the ideal initial conditions scenario with random initial conditions, it failed to complete a single sprint in 54% of its trials, and it was so prone to collisions that it never reached a fitness score above 0.1784 (22 collisions).

7. CONCLUSIONS AND FUTURE WORK

In this paper, we described a design process for engineering a self-organizing box-pushing system, whose agents have basic hardware and are not capable of system-wide synchronization. Agent behavior relied on state switching between forming and pushing, with the pushing governed by agents' positions within a tField, and the forming governed by a combined sField and tField. Using agent-based simulation and a GA, we were able to explore the tradeoffs that designers of self-organizing systems must make between performance and adaptability. We found that if a system is optimized for robustness, by allowing it to evolve in the GA in the face of random initial conditions and perturbations, it will display this robustness in future testing, but its maximum values may be attenuated. We also found that great performance gains can come from control over the system's initial conditions, but that a system optimized with ideal initial conditions risks catastrophic failure if deployed in uncertain environments due to its lack of robustness.

Limitations

The quantitative findings presented here rest on the assumption that the GA was able to find the optimal parameter set for every scenario and behavior encoding. As with any heuristic search algorithm, this cannot be guaranteed. Given the huge search space, it would be impossible to perform an exhaustive search to prove optimality, but these are the best candidates that could be found within the time and computing power constraints that we faced.

The simulation physics could be made more realistic. The physical model is still first-order, with forces corresponding directly to movements. A more realistic model would have forces creating accelerations, which would complicate the dynamics of the system, and would require a more sophisticated model of friction. The collision detection is also not foolproof. This is due to NetLogo's assumptions, which models all moving entities as points with a surrounding circle. This can only approximately represent the rectangular box, and a large number of small circles is used in this simulation to "fill in" the box.

Future Work

The major arc of this research is to develop a methodology for conceptual modeling of self-organizing systems. How to form the parametric behavioral models in this and our previous work is an open question, but this work is an attempt to understand some of the tradeoffs that a designer has to face. Present work in our laboratory also has the goal of transferring the behavioral algorithms listed here onto physical robots, for testing on real hardware rather than solely in simulation.

REFERENCES

[1] R. Neches and A. M. Madni, "Towards affordably adaptable and effective systems," *Systems Engineering*, p. n/a–n/a, 2012.

[2] A. B. Urken, A. "Buck" Nimz, and T. M. Schuck, "Designing evolvable systems in a framework of robust, resilient and sustainable engineering analysis," *Advanced Engineering Informatics*, vol. 26, no. 3, pp. 553–562, Aug. 2012.

[3] A. Requicha, "Swarms of Self-Organized Nanorobots," in *Nanorobotics*, Springer, 2013, pp. 41–49.

[4] T. Hogg, "Distributed control of microscopic robots in biomedical applications," in *Advances in applied self-organizing systems*, Springer, 2013, pp. 179–208.

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

[6] S. Kornienko, O. Kornienko, A. Nagarathinam, and P. Levi, "From real robot swarm to evolutionary multi-robot organism," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, 2007, pp. 1483–1490.

[7] C. J. Sanchez, C.-W. Chiu, Y. Zhou, J. M. Gonzalez, S. B. Vinson, and H. Liang, "Locomotion control of hybrid cockroach robots," *Journal of The Royal Society Interface*, vol. 12, no. 105, pp. 20141363–20141363, Mar. 2015.

[8] J. C. Nawroth, H. Lee, A. W. Feinberg, C. M. Ripplinger, M. L. McCain, A. Grosberg, J. O. Dabiri, and K. K. Parker, "A tissue-engineered jellyfish with biomimetic propulsion," *Nature Biotechnology*, vol. 30, no. 8, pp. 792–797, 2012.

[9] H. Simon, "The Architecture of Complexity," *Proceedings of the American Philosophical Society*, vol. 106, no. 6, pp. 467–482, 1962.

[10] Y. Bar-Yam, A. Minai, and D. Braha, "Complex Engineered Systems: A New Paradigm," in *Complex engineered systems: science meets technology*, Berlin ; New York: Springer, 2006, pp. 23–39.

[11] N. P. Suh, "A Theory of Complexity, Periodicity and the Design Axioms," *Research in Engineering Design*, vol. 11, no. 2, pp. 116–132, Aug. 1999.

[12] N. Khani and Y. Jin, "Dynamic Structuring in Cellular Self-Organizing Systems," in *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, Eds. Cham: Springer International Publishing, 2015, pp. 3–20.

[13] N. Khani, J. Humann, and Y. Jin, "Effect of Social Structuring in Self-Organizing Systems," *Journal of Mechanical Design*, vol. 138, no. 4, 2016.

[14] S. Camazine, *Self-organization in biological systems*. Princeton, N.J.: Princeton University Press, 2001.

[15] V. Trianni, *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots*. Berlin: Springer, 2008.

[16] C. Jones and M. J. Matarić, "Adaptive division of labor in large-scale minimalist multi-robot systems," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2003, vol. 2, pp. 1969–1974.

[17] J. Buhl, D. J. T. Sumpter, I. D. Couzin, J. J. Hale, E. Despland, E. R. Miller, and S. J. Simpson, "From Disorder to Order in Marching Locusts," *Science*, vol. 312, no. 5778, pp. 1402–1406, Jun. 2006.

[18] C. R. Kube and E. Bonabeau, "Cooperative transport by ants and robots," *Robotics and autonomous systems*, vol. 30, no. 1, pp. 85–101, 2000.

[19] E. Bonabeau, *Swarm intelligence: from natural to artificial systems*. New York: Oxford University Press, 1999.

[20] E. Bonabeau, G. Theraulaz, J. Deneubourg, N. R. Franks, O. Rafelsberger, J. Joly, and S. Blanco, "A model for the emergence of pillars, walls and royal chambers in termite nests," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 353, no. 1375, pp. 1561–1576, 1998.

[21] J. Korb, "Termite Mound Architecture, from Function to Construction," in *Biology of Termites: a Modern Synthesis*, D. E. Bignell, Y. Roisin, and N. Lo, Eds. Springer Netherlands, 2011, pp. 349–373.

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

[23] 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.

[24] M. Moussaïd, D. Helbing, and G. Theraulaz, "How simple rules determine pedestrian behavior and crowd disasters," *PNAS*, vol. 108, no. 17, pp. 6884–6888, Apr. 2011.

[25] D. J. Nowak, G. B. Lamont, and G. L. Peterson, "Emergent architecture in self organized swarm systems for military applications," in *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, New York, NY, USA, 2008, pp. 1913–1920.

[26] P. Dasgupta, "A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 38, no. 3, pp. 549–563, 2008.

[27] F. Ruini and A. Cangelosi, "Extending the Evolutionary Robotics approach to flying machines: An application to MAV teams," *Neural Networks*, vol. 22, no. 5, pp. 812–821, 2009.

[28] D. Zhang, L. Wang, and J. Yu, "A Coordination Method for Multiple Biomimetic Robotic Fish in Underwater Transport Task," in *American Control Conference, 2007. ACC'07*, 2007, pp. 1870–1875.

[29] Y. Hu, L. Wang, J. Liang, and T. Wang, "Cooperative box-pushing with multiple autonomous robotic fish in underwater environment," *IET control theory & applications*, vol. 5, no. 17, pp. 2015–2022, 2011.

[30] E. F. Parra-González, G. Ramírez-Torres, and G. Toscano-Pulido, "Motion planning for cooperative multi-robot box-pushing problem," in *Advances in Artificial Intelligence–IBERAMIA 2008*, Springer, 2008, pp. 382–391.

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

[32] 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.

[33] J. Werfel, "Collective Construction with Robot Swarms," in *Morphogenetic Engineering: Toward Programmable Complex Systems*, 2012, pp. 115–140.

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

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

[36] Y. Jin and C. Chen, "Cellular self-organizing systems: A field-based behavior regulation approach," *AI EDAM*, vol. 28, no. Special Issue 02, pp. 115–128, May 2014.

[37] J. Humann and A. M. Madni, "Integrated Agent-based modeling and optimization in complex systems analysis," *Procedia Computer Science*, vol. 28, pp. 818–827, 2014.

[38] J. Humann, Y. Jin, and N. Khani, "Evolutionary Computational Synthesis of Self-Organizing Systems," *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2014.

[39] M. W. Moffett, "Cooperative food transport by an Asiatic ant," *National Geographic Research*, vol. 4, no. 3, pp. 386–394, 1988.

[40] S. M. LaValle, *Planning algorithms*. Cambridge ; New York: Cambridge University Press, 2006.

[41] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem. II. General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, no. 3, pp. 298–351, Sep. 1983.

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

[43] 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.

[44] J. McLurkin, "Experiment design for large Multi-Robot systems," in *Robotics: Science and Systems, Workshop on Good Experimental Methodology in Robotics, Seattle, WA, USA*, 2009.

[45] A. Wagner, *Robustness and evolvability in living systems*. Princeton, N.J: Princeton University Press, 2005.

[46] C. Gershenson, "A general methodology for designing self-organizing systems," *arXiv preprint nlin/0505009*, 2005.

[47] T. De Wolf and T. Holvoet, "Towards a methodology for engineering self-organising emergent systems," *Frontiers in Artificial Intelligence and Applications*, vol. 135, p. 18, 2005.

[48] S. Pugh, *Total design*. Addison-Wesley, 1990.

[49] M. W. Maier and E. Rechtin, *The art of systems architecting*, 3rd ed. Boca Raton: CRC Press, 2009.

[50] J. Humann and Y. Jin, "Evolutionary Design of Cellular Self-Organizing Systems," in *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2013, pp. V03AT03A046–V03AT03A046.

[51] B. Edmonds, "Using the experimental method to produce reliable self-organised systems," in *Engineering Self-Organising Systems*, Springer, 2005, pp. 84–99.

[52] J. Schellinck and T. White, "A review of attraction and repulsion models of aggregation: Methods, findings and a discussion of model validation," *Ecological Modelling*, vol. 222, no. 11, pp. 1897–1911, 2011.

[53] W. Chiang and Y. Jin, "Toward a Meta-Model of Behavioral Interaction for Designing Complex Adaptive Systems," in *ASME IDETC/CIE 2011*, 2011, pp. 1077–1088.

[54] 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.

[55] J. S. Gero and U. Kannengiesser, "The situated function–behaviour–structure framework," *Design Studies*, vol. 25, no. 4, pp. 373–391, Jul. 2004.

[56] D. Goldberg, *The Design of Innovation*, 1st ed. Springer, 2002.

[57] W. M. Spears and V. Anand, *A study of crossover operators in genetic programming*. Springer, 1991.

[58] A. Agah and G. A. Bekey, "A genetic algorithm-based controller for decentralized multi-agent robotic systems," in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, 1996, pp. 431 –436.

[59] 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.

[60] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[61] S. Luke, "The ECJ Owner's Manual," *Department of Computer Science, George Mason University, zeroth edition*, 2010.

[62] B. Birge, "Applying Biomimetic Algorithms for Extra-Terrestrial Habitat Generation," Aug. 2012.

[63] M. Mamei and F. Zambonelli, "Theory and practice of field-based motion coordination in multiagent systems," *Applied Artificial Intelligence*, vol. 20, no. 2–4, pp. 305–326, 2006.