Scalability in Self-Organizing Systems: An Experimental Case Study on Foraging Systems

Chapter · January 2018

DOI: 10.1007/978-3-319-62217-0_38

3 authors:

James Humann
Army Research Laboratory
10 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Yan Jin
University of Southern California
98 PUBLICATIONS **2,050** CITATIONS
SEE PROFILE

Azad M. Madni
University of Southern California
186 PUBLICATIONS **1,152** CITATIONS
SEE PROFILE

Chapter 38
Scalability in Self-Organizing Systems: An Experimental Case Study on Foraging Systems

James Humann, Yan Jin, and Azad M. Madni

Abstract Scalability is a great advantage for systems that face uncertain demand. Scalable systems can be increased in size at a reasonable cost to meet increasing demand, or they can be reduced in size to minimize ongoing costs in the face of falling demand. Self-organization is often hailed as a strategy for creating scalable systems, as they have low integration costs and no communication bandwidth limit from a central controller. This paper investigates the scalability of a self-organizing foraging system. The results show that there are fitness penalties associated with scaling systems up or down from the size they had been optimized for, and these penalties are higher for scaling up rather than down. However, if the system's agent behavioral parameters can be adjusted as the system size changes, the system-level fitness increases linearly with size.

Keywords Scalability • Self-organization • Agent-based modeling • Multi-agent simulation

38.1 Introduction

38.1.1 Scalability

Dynamic allocation of resources is an important strategy in uncertain environments. Scalable systems can change in size to meet changing requirements [1], grow when they face greater demand, and shrink to meet falling demand with fewer resources. A rigid system that is not scalable may incur costs if it is undersized or oversized. These costs include the excessive costs of building a system's capabilities beyond their demanded performance, ongoing maintenance of a larger system, or opportunity cost of not capitalizing on higher than expected demand. This places enormous

J. Humann, PhD (✉)
Intelligent Systems Technology, Inc., Los Angeles, CA, USA
e-mail: humann@usc.edu

Y. Jin, PhD • A.M. Madni, PhD
University of Southern California, Los Angeles, CA, USA
e-mail: yjin@usc.edu; azad.madni@usc.edu

A.M. Madni et al. (eds.), *Disciplinary Convergence in Systems Engineering Research*, https://doi.org/10.1007/978-3-319-62217-0_38

pressure on the system's engineer to design the system's capacity so that it is just right, but this may not be a realistic expectation for complex environments or long-life-span systems.

38.1.2 Self-Organizing Systems

Self-organization is one strategy for designing systems that can scale to their required capacity at run time. A self-organizing system is made of a group of interacting autonomous agents that are not subject to any central controller. Integration costs in self-organizing systems are low, so adding capacity is only limited by the cost of the hardware. Their distributed nature also eliminates a possible upper limit on system size that could be a constraint of a central controller. An inspiration from the natural realm, locust swarms, can grow in size up to 10^9 insects [2] yet still fly as a cohesive flock without any single locust leading or coordinating the swarm. Two complementary forces have recently increased the importance and visibility of self-organized architectures: a market pull (from customers requiring adaptability, scalability, and resilience in systems) and a technology push (from enabling technologies such as miniaturized robots, bio-inspired robotics, and the Internet of Things) [3].

Self-organizing systems rely on local communication and sensing, so their control schemes have inherent potential for scalability. A centralized controller would have to read and synthesize all of the data available from lower-level sensors [4]. As the size of the system increases, so do the demands on the central controller until it is pushed beyond its capacity, which would cause it to crash or at best delay its outputs or ignore some information. In self-organizing systems, agents form dynamic local networks that are relatively stable in size and may be loosely linked to one another. As long as the agents are properly designed to analyze the data available to them locally, more can be added to the system without stressing a system-wide bandwidth limitation. Despite the local frame of action, the interactions among agents can spread information system-wide and enable complex system-level behavior [5]. Thus scalability is often hailed as a promising feature of self-organizing systems [5, 6], but naively scaling systems without understanding the possible pitfalls may lead to system failures [7, 8].

38.1.3 Related Work

Ross et al. give a formal, domain-independent definition of scalability as a specific type of changeability, among other "ilities" such as adaptability and modifiability [1]. In [5, 9] it was shown that large scalable vehicle networks could be formed from peer-to-peer communication. This network could disseminate safety and

traffic information orders of magnitude further than the peer-to-peer transmission range without relying on or being limited by any third-party infrastructure.

Self-organization has been suggested as a strategy for scalable formation control for pedestrians and vehicles [10, 11]. In fact, the Boids algorithm, a famous self-organized flocking algorithm, became popular initially because it was so computationally efficient in its ability to display computer-generated flocks of birds [12]. Self-organized foraging in robotic systems has been demonstrated in [7, 13] where groups of robots gathered pucks and boxes into a central location without directly communicating.

This paper is also the continuation of a series on cellular self-organizing (CSO) systems, so called because each agent in the system is rather simple, but by working together, the agents can display complex behavior, like the cells in a human body. CSO research has demonstrated reconfigurability through information sharing [14], field-based control for searching and swarm formation [15, 16], and applications in exploration, box pushing, and protective tasks [17, 18]. The two complementary goals of CSO research are to understand self-organization in natural systems and to apply this knowledge to the design of engineered resilient systems [19].

38.1.4 Agent-Based Modeling and Genetic Algorithms

In [20, 21], we introduced a methodology for the design of self-organizing systems. The methodology relies heavily on the use of agent-based modeling for system analysis and genetic algorithms for optimization. Agent-based modeling treats elements of the system as autonomous actors with sensing, reasoning, and decision-making capabilities. The interactions of the agents can be simulated and tested on a computer to study emergent properties of the system. This approach allows the engineer to focus on a small-scale problem: accurately defining agent behavior. The larger-scale problem, determining the complex results of agent interactions, is left to the computer. Genetic algorithms (GA) have been used in previous work [22, 23] on the design of multi-agent systems because they are efficient algorithms for optimizing large, complex, and noisy search spaces. Briefly, a genetic algorithm operates on a population of potential solutions (agent behavior settings) by scoring them with a fitness functions and improving them in successive generations [24, 25].

Combining the two approaches allows a designer to focus on the conceptual design of the agent behavior. As long as it is parameterized, creating a class of systems, the genetic algorithm can search for an optimal point solution by repeatedly invoking the multi-agent simulation software.

38.1.5 The Foraging Task

In a foraging system, agents must find a resource and transport it back to a base location. One of the most famous examples is found in nature: the food foraging behavior of ants [26, 27]. Ants begin searching for food around their nest individually. When one by chance finds food, it lays down a specific pheromone (chemical scent) as it carries the food back to the nest. Other ants then randomly find the pheromone trail, follow it to the food source, and lay down even more pheromones in the same location, causing a positive feedback loop of increasing pheromone concentration. Eventually, the whole colony is recruited to exploit the food source, forming an efficient straight line between the food and the nest. This ant behavior is actually so adept at solving search problems that have been abstracted into an optimization algorithm, known as ant colony optimization [28].

In this paper, the foraging behavior of ants serves as an inspiration for the design of an artificial self-organized foraging system. In practical applications, the basic pattern could be seen in a system that finds and gathers waste in a cleanup task, a search and rescue system, or a system that harvests crops.

38.2 Experimental Setup

38.2.1 Foraging Task and Simulation

Figure 38.1 shows the initial setup of the simulated foraging task in NetLogo [29]. The food is marked in green, and the home base is marked in red. The objective is to maximize the amount of food returned to home within a time limit. Agents can sense food and other agents within 3 pw (pw is the width of a "patch" in the simulation world; the size of a patch can be seen in the blocks forming the home base). When an agent moves onto a patch that contains food, it extracts five units of food and changes its own color to green, signifying to other agents that it has found food. If it carries food back to the home base, it deposits the food and changes its color back to brown. This stored food then counts toward the system's fitness score. Agents maintain no memory of the food location and must find it anew every time they leave the home base. They can only sense it when it is within their 3 pw detection radius. Agents can, however, sense the direction toward home at all times.

38.2.2 Agent Behavioral Model

The agents in this system have two states: carrying food or not carrying food. They have 18 state-based behavioral parameters, summarized in Table 38.1.

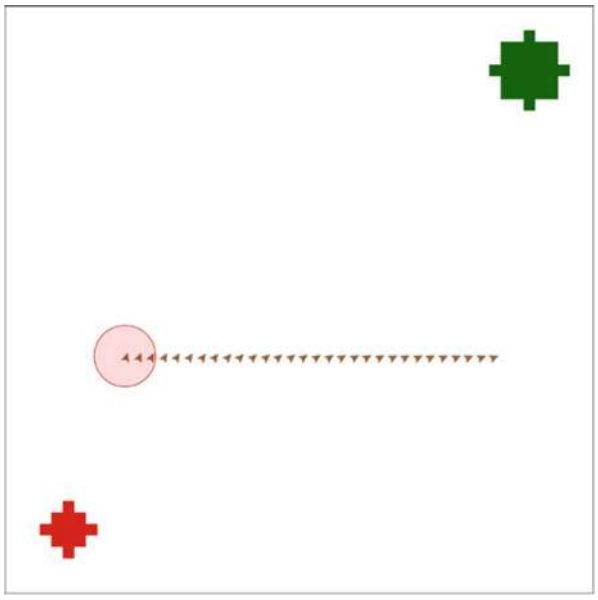

Fig. 38.1 Initial configuration of a one-row foraging simulation. The *red circle* indicates the detection range of an agent

Table 38.1 Foraging behavioral parameters

Agent state	Neighbor state	Cohesion	Avoidance	Alignment	Randomness	Home	Food
Food	Food	C_1	O_1	A_1	R_1	H_1	F_1
	No food	C_2	O_2	A_2			
No food	Food	C_3	O_3	A_3	R_2	H_2	F_2
	No food	C_4	O_4	A_4			

Notionally, cohesion is an agent's desire to move toward its neighbors; avoidance is its desire to move away from its neighbors; alignment is its desire to match speed and direction with its neighbors; the randomness desire changes at each time step; home is the desire to move toward the home base; and food is the desire to move toward sensed food. These desires are all considered simultaneously, and a weighted average of the stimuli is used by the agent to decide on its next step.

To aid in this decision process, field-based behavior regulation [19, 22] is used. Field-based regulation treats all stimuli as sources or sinks in a mathematical field. In this paper, agents consider two fields: a task field of stimuli in the environment and task and a social field of other agents. This separation is used to aid the designer, as the social field can create system structure while the task field deploys it in space. Agents calculate the field value at every reachable point in their immediate vicinity and step to the point with the highest field value. The field equations are given as

$$FLD_{\mathrm{S}}(r,\ \theta,\ \phi) = C \cdot \frac{-1}{N}\sum_{i\in\eta}^{N} r_i + O \cdot \frac{-1}{N}\sum_{i\in\eta}^{N}\frac{1}{r_i} + A \cdot \frac{1}{N}\sum_{i\in\eta}^{N}|v_i|\ \cos(\theta-\phi) \tag{38.1}$$

$$\mathrm{FLD_t}(f,\ h,\ \phi) = S_{\max} \cdot F \cdot (\cos(f-\phi) + H \cdot \cos(h-\phi)) \tag{38.2}$$

where η is the set of N agents in the calculating agent's radius of detection; r_{i} is the distance from a point to the agent's neighbor; ϕ is the point's angle away from the agent's current heading; θ is the neighbor's current heading relative to the agent; $s_{\max}$ is the agent's maximum step size; f is the angle toward food; h is the angle toward home; and C, O, A, F, and H are state-determined parameters as described in Table 38.1. The results of Eqs. (38.1) and (38.2) are added together to calculate the field value of any point.

38.2.3 Simulation and Optimization

The 18 behavioral parameters given in Table 38.1 define a class of systems. Any set of particular parameters fixes the behavior of that system. This allows the GA to search through a space of possible systems for optimal behavior. The GA's fitness function is given as

$$\text{fitness} = \text{food}_{\mathrm{r}} + \frac{1}{N}\sum_{i=1}^{N}\text{food}_{\mathrm{c},i} \tag{38.3}$$

where food_{r} is the food returned by the end of the simulation and food_{c} is the food being carried by agents (but not yet returned) at the final time step. The summation is carried out over all N agents in the simulation.

At each time step, every agent will sense its local neighborhood and apply its behavioral algorithm. If an agent finds a patch containing food, it picks up and begins carrying five units of food. It carries the food to the home base, it drops the food, and the food then counts toward the system's total fitness. This is repeated for 1000 time steps, and the systems are judged at the end according to Eq. (38.3).

38.2.4 Scalability Assessment

To test for scalability, behavioral parameters are optimized and tested for each of one to six rows of agents. Then, keeping the parameters constant, the systems are tested in other scenarios. In this way, each system is tested both within and outside

of the nominal size for which it was optimized. If increasing size leads to improved performance, we can call the system scalable. Because self-organizing systems have low integration costs, the majority of the cost of increasing system size comes from the actual agent hardware. If the performance of the system increases proportionally to the cost, we can call it linearly scalable. System scalability can also be superlinear if the performance increases with the number of agents and the rate of increase also increases. Sublinear systems get diminishing returns from adding agents, and in the worst case, their performance may even deteriorate.

38.2.5 Extended Optimization

The investigation in this paper requires that an optimal candidate be found for each scenario. In a complex system, it is very difficult to objectively determine which solution set is optimal for several reasons. The search space is enormous, and the simulations are partially stochastic, so to get statistical confidence, many trials would have to be performed. The GA is also partially stochastic, and different GA runs may converge to different parameter sets even if they are optimizing within the same search space. So in this paper, we do not refer to optimal candidates, but instead optimized candidates.

A consistent process was used to generate optimized candidates in each scenario, enabling a fair comparison of performance across scenarios. The process includes five GA runs. The three best candidates found from each of the first four runs are used to seed the fifth. The 15 best candidates (3 from each run) are then retested for reliability by evaluating their fitness in 100 simulations. The optimized candidate is chosen as the one with the highest 30th percentile performance out of these 100 runs.

38.3 Results and Implications

Note: the results show a distinction between systems *optimized for* a certain size and systems *deployed at* a certain size. In text, *RO-X* will denote the size a system is optimized for (where X represents the number of agent rows), and an *X-row* system will refer to the actual number of rows in deployment. For example, an RO2 six-row system refers to a system deployed with six rows of agents whose behavior was optimized for two rows of agents (Figs. 38.2, 38.3, 38.4, and 38.5).

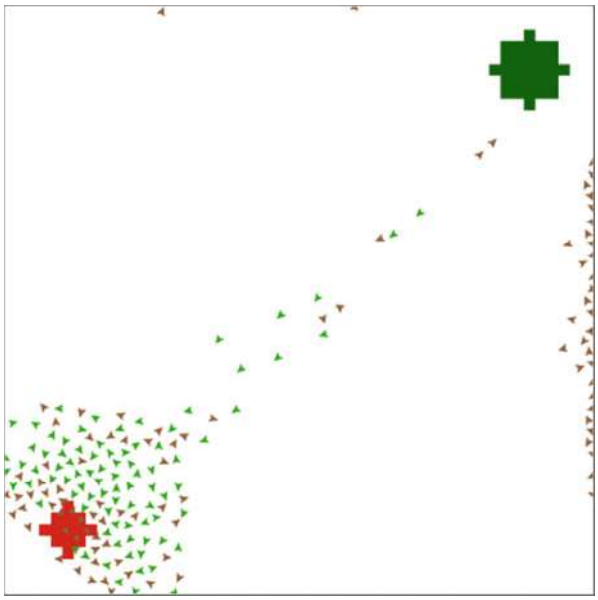

Fig. 38.2 RO1 six-row system showing jamming around home base

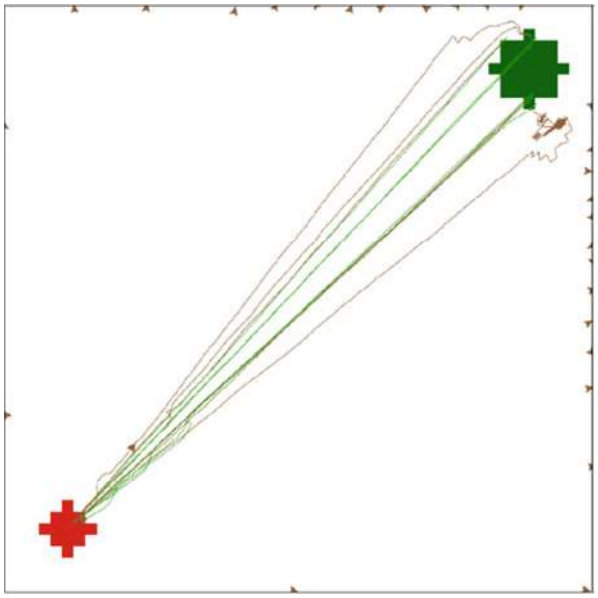

Fig. 38.3 Behavior of RO6 one-row system for time steps 500–1000

38.3.1 Scalability of Conceptual Design

Figure 38.5 shows the fitness of the optimized systems for each of one-to-six-row systems, after optimizing the parameters of Table 38.1 at each size. This

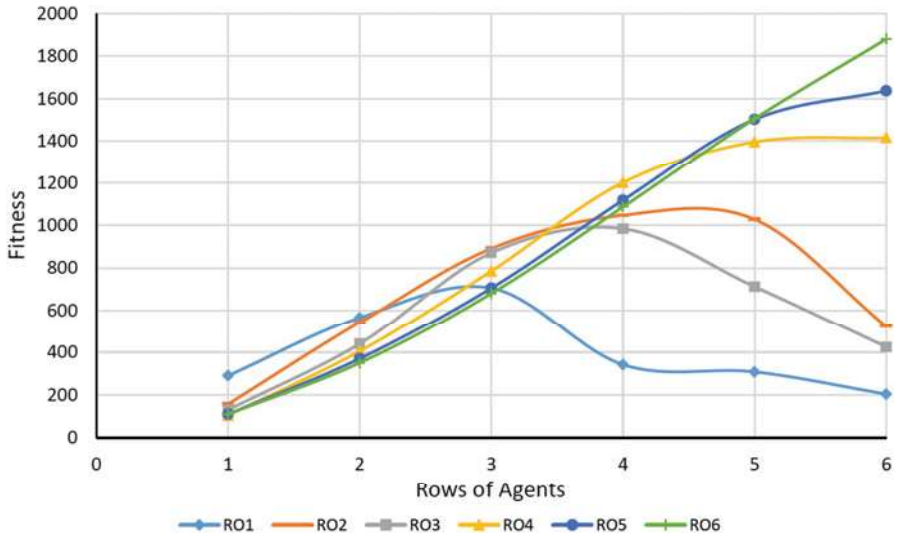

Fig. 38.4 Results of scalability test, where each curve represents one set of behavioral parameters, tested at each system size

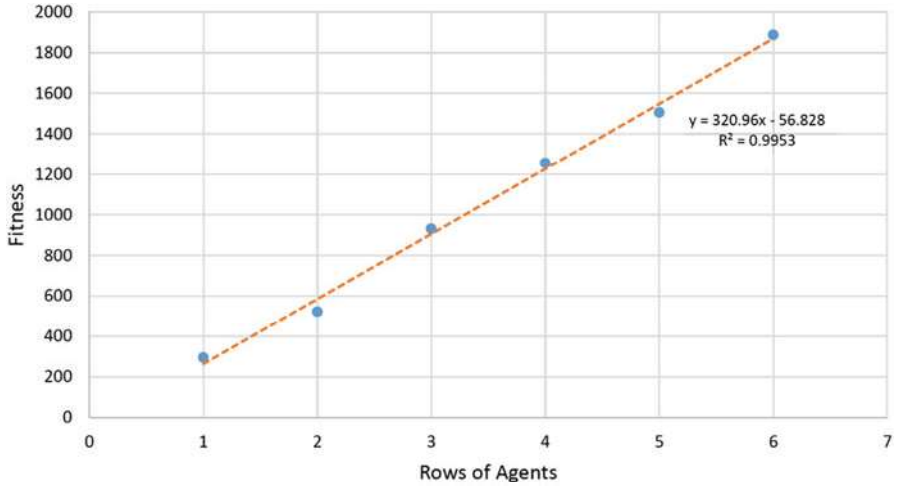

Fig. 38.5 Optimized fitness for each number of agent rows, showing a linear relationship between performance and system size

corresponds to the case where the designer can change behavioral parameters (i.e., change the detail design while maintaining the conceptual design) as more agents are added to the system. As seen in the figure, the system is almost perfectly linearly scalable. The R^2 value for the linear regression is 0.995.

38.3.2 Scalability of Detail Design

What can be done if the system operator is not allowed to change behavioral design parameters after design or deployment? In such a scenario, a system with parameters optimized for one row of agents may be scaled to six rows or vice versa. This constraint was explored by taking the optimized parameter sets from Sect. 38.3.1 and testing them for each of one to six rows of agents. Thus they were sometimes tested outside of the size range for which they were optimized. The results are summarized in Fig. 38.4.

It can be seen that systems optimized for small sizes (RO1, RO2, RO3) were unable to effectively scale up in size, but the systems optimized for large sizes were able to smoothly scale down in size. This implies that there is some directionality in the conceptual design, making it easier to decrease in size than to increase if the individual agent behavior is held constant. At either end, there was a fitness penalty for deploying a system at a given size larger or smaller than its optimized size, when compared with a system optimized for that size. (Note that due to small deficiencies in the partially stochastic optimization, there were several cases where this is not true. The most notable is two-row systems, where RO1 outperforms RO2 by 4.4%. All other cases manifest by a slimmer margin or have the expected system performing the best.) Table 38.2 shows the fitness penalties for cross-testing the extreme ends of the size range.

The most drastic fitness penalty comes from scaling the RO1 system to a six-row system. Figure 38.2 shows the end state of this scenario: the agents carrying food crowded around the home base and jammed the system. The reversed scenario (RO6 one row) also showed interesting results with a high fitness penalty. The strategy chosen by the GA during optimization (for a six-row system) was for the system to leave a layer of agents stuck along the boundary of the field, guiding a circulating inner core to find and retrieve food. This behavior is shown in Fig. 38.6. Contrast that figure with Fig. 38.3, which shows the RO6 one-row system attempting to exploit the same strategy, but it leaves too high a fraction of its agents static along the wall. With so few agents doing the foraging required to raise its fitness score, it suffers a 62% fitness penalty compared to the RO1 system.

Table 38.2 Results of cross-testing systems optimized for large size in small-scale deployment and vice versa

System size (rows)	Optimized fitness	Test system	Test fitness	Penalty (%)
1	290.0	RO5	110.0	62.1
		RO6	110.2	62.0
2	542.1	RO5	370.3	31.7
		RO6	350.3	35.4
5	1502.5	ROl	308.0	79.5
		RO2	1027.6	31.6
6	1881.1	ROl	204.0	89.2
		RO2	525.2	72.1

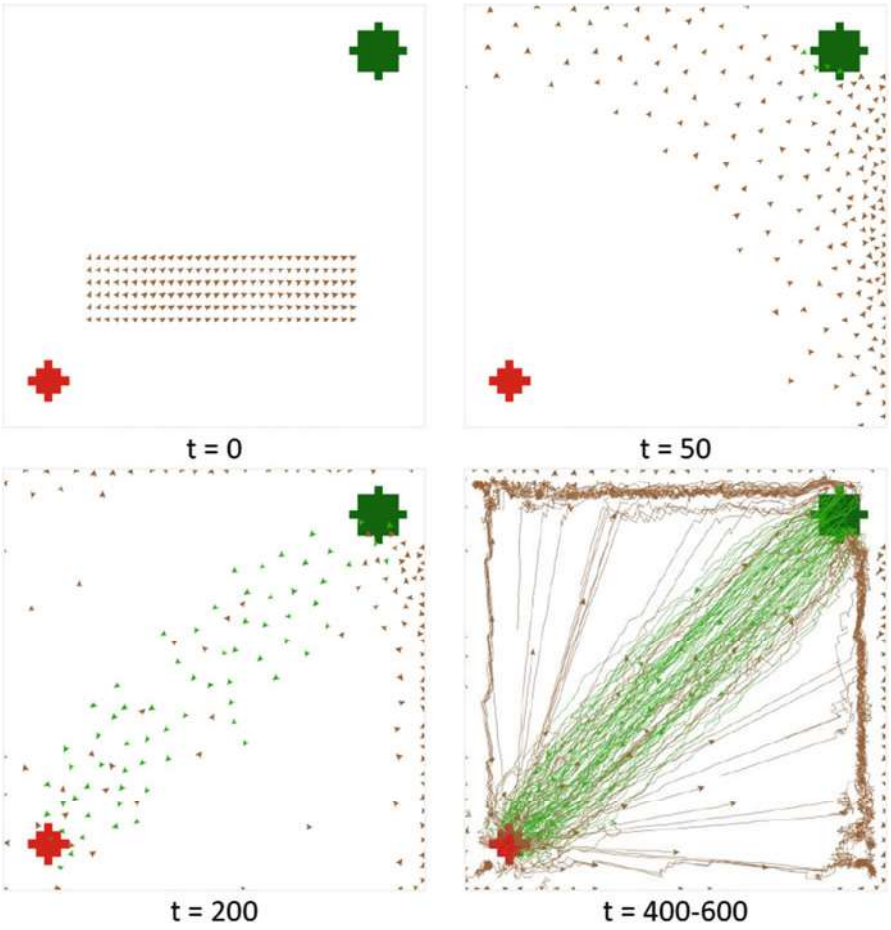

Fig. 38.6 Behavior of RO6 six-row system. The fourth panel shows a motion trace of each agent for 200 time steps

38.3.3 Scalability of System with Boundary Detection

It was shown in [22] that when agents are able to detect and react to the boundary of the field, they can flock and move much more smoothly, returning food more efficiently. This adds two behavioral parameters to the set from Table 38.1. A similar test was carried out for systems with boundary detection, and the results are shown in Fig. 38.7 and Table 38.3. The results follow the pattern of the previous section but are more extreme. There were fitness penalties up to 99% when scaling a system up in size and milder penalties when decreasing system size. The conceptual design was again linearly scalable ($R2 = 0.999$).

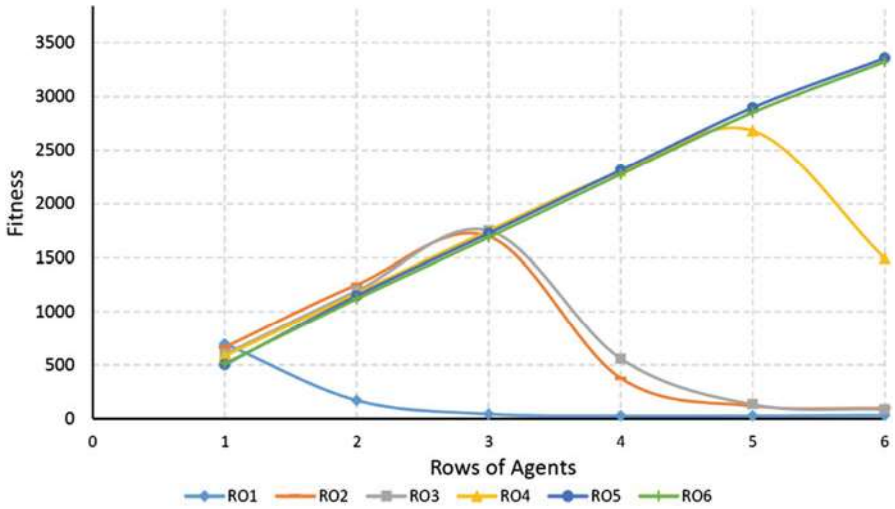

Fig. 38.7 Scalability tests for systems with boundary detection

Table 38.3 Results of cross-testing systems with boundary detection optimized for large size in small-scale deployment and vice versa

System size (rows)	Optimized fitness	Test system	Test fitness	Penalty (%)
1	697.4	RO5	502.3	28.0
		RO6	511.8	26.6
2	1249.1	RO5	1147.0	8.17
		RO6	1117.5	10.5
5	2896.7	RO1	25.75	99.1
		RO2	117.5	95.9
6	3323.3	RO1	30.45	99.1
		RO2	97,4	97.1

38.4 Discussion

At the conceptual design level, both sets of experiments showed linear scalability. For systems engineers, this means that scalable foraging systems are feasible, as long as there is a way to adjust the parameters according to the system size. Since the behavior is primarily dominated by software and parameters, this should not incur major cost.

However, the results show that it is difficult to scale a self-organizing system up in size if the behavioral parameters cannot be adjusted accordingly. Not only did the RO1 and RO2 systems suffer a relative fitness penalty as five-row and six-row systems compared to the RO5 and RO6 systems, but above a certain size, they even lose fitness in absolute terms. Looking at the simulation results qualitatively, agent groups cause jamming around the food or home base, halting progress early in the simulation run.

Scaling the RO5 and RO6 systems down in size was seen to be less problematic, although the relative fitness penalties were as high as 62%. The problem was that the behaviors and structures selected by the GA were quite effective at large sizes, but did not have the critical mass to be effective at small sizes. For example, in the RO6 six-row system without boundary detection, a small fraction of the agents could be sacrificed along the boundary in order to guide the rest of the agents toward the food, but as a one-row system, the number of static agents required for this strategy caused too much relative overhead and seriously limited the speed at which the system could find and return food.

38.5 Conclusion

38.5.1 Summary and Conclusions

With increasing connectivity and miniaturization of robots, there is greater opportunity for engineers to design distributed systems with self-organized architectures. The advantages of these systems are redundancy, adaptability, mass production, and possible scalability. Scalability was shown in this paper to be dependent on behavior parameters chosen during optimization. Systems optimized for small size suffered from jamming at large sizes, and systems optimized for large sizes did not have the resources to form large-scale subsystems at small sizes. The methodology, based on agent-based simulation, parametric behavioral modeling, and genetic optimization, was shown to effectively uncover these strategies and possible pitfalls. The engineer's responsibility is to leverage this information, knowledge of the system's environment, and use cases to tailor the agents' behavior to the appropriate size or size range.

38.5.2 Limitations and Future Work

The results of this paper are specific to the foraging simulations in question. They are meant to serve as data points in the study of self-organizing systems, and care must be taken in transferring specific results (e.g., fitness penalties) to the design of scalable complex systems in other domains. Nonetheless, the qualitative lessons, scalability assessment, and design methodology are generalizable to the design of many different self-organizing systems (see [21] for related examples). The simulation/optimization was also limited by the computational power available, as an exhaustive search of the 18-parameter space was impractical. With increased computational ability, the optimized systems will be closer to the true optimum.

For future work, we will look into various methods for adapting the behavior of the system to its specific size. This raises interesting questions of self-knowledge

(how does a distributed system know its own size?) and how to distribute behavioral updates (should a central controller broadcast updates, or should they spread virally?). There are ongoing efforts to transfer the behavioral models in this research to physical robots, instead of just in simulation. Also, the general lessons on self-organization continue to inform our ongoing research on groups of autonomous ground, sea, and air vehicles [30, 31].

References

1. Ross AM, Rhodes DH, Hastings DE (2008) Defining changeability: reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. Syst Eng 11(3):246–262
2. Buhl J et al (2006) From disorder to order in marching locusts. Science 312(5778):1402–1406
3. Humann J, Khani N, Jin Y (2016) Adaptability tradeoffs in the design of self-organizing systems. In: Proceedings of the ASME 2016 IDETC/CIE Conference, Charlotte
4. Matni N, Leong YP, Wang YS, You S, Horowitz MB, Doyle JC (2014) Resilience in large scale distributed systems. Procedia Comput Sci 28:285–293
5. Wischhof L, Ebner A, Rohling H (2005) Information dissemination in self-organizing intervehicle networks. IEEE Trans Intell Transp Syst 6(1):90–101
6. Dorigo M et al (2004) Evolving self-organizing behaviors for a swarm-bot. Auton Robot 17 (2–3):223–245
7. Beckers R, Holl OE, Deneubourg JL, Bielefeld Z, Bielefeld D (1994) From local actions to global tasks: stigmergy and collective robotics. In: Artificial Life IV. MIT Press, Cambridge, MA, pp 181–189
8. Petroski H (1994) Design paradigms: case histories of error and judgment in engineering. Cambridge University Press, Cambridge/New York
9. Bauza R, Gozálvez J (2013) Traffic congestion detection in large-scale scenarios using vehicle-to-vehicle communications. J Netw Comput Appl 36(5):1295–1307
10. Mikhailov AS (2011) From Swarms to Societies: Origins of Social Organization. In: Meyer-Ortmanns H, Thurner S (eds) Principles of Evolution. Springer Berlin Heidelberg, Berlin/Heidelberg, pp 367–380
11. Nowak DJ, Lamont GB, Peterson GL (2008) 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, pp 1913–1920
12. Reynolds CW (1987) Flocks, herds, and schools: a distributed behavioral model. In: ACM SIGGRAPH'87 conference proceedings, vol 25–34. Anaheim, CA
13. Song Y, Kim J-H, Shell D (2012) Self-organized clustering of square objects by multiple robots. In: Dorigo M, Birattari M, Blum C, Christensen A, Engelbrecht A, Groß R, Stützle T (eds) Swarm intelligence, vol 7461. Springer, Berlin/Heidelberg, pp 308–315
14. Zouein G, Chen C, Jin Y (2010) Create adaptive systems through 'DNA' guided cellular formation. In: Design creativity 2010. Springer, p 149
15. Jin Y, Chen C (2014) Cellular self-organizing systems: a field-based behavior regulation approach. Artif Intell Eng Des Anal Manuf 28(2):115–128
16. Chiang W, Jin Y (2011) Toward a meta-model of behavioral interaction for designing complex adaptive systems In: ASME IDETC/CIE 2011, pp 1077–1088
17. Khani N, Humann J, Jin Y (2016) Effect of social structuring in self-organizing systems. J Mech Des 138(4):041101

18. Humann J, Jin Y (2013) Evolutionary design of cellular self-organizing systems. In: ASME 2013 international design engineering technical conferences and computers and information in engineering conference, pp V03AT03A046–V03AT03A046
19. Jin Y, Chen C (2012) Field based behavior regulation for self-organization in cellular systems. In: Presented at the Design Computing and Cognition Conference DCC'12, 2012
20. Humann J, Madni AM (2014) Integrated agent-based modeling and optimization in complex systems analysis. Procedia Comput Sci 28:818–827
21. Humann J (2015) Computational synthesis and behavioral modeling of self-organizing systems. PhD, University of Southern California, Los Angeles
22. Humann J, Khani N, Jin Y (2014) Evolutionary computational synthesis of self-organizing systems. AI EDAM 28(03):259–275
23. Calvez B, Hutzler G (2006) Automatic tuning of agent-based models using genetic algorithms. In: Sichman J, Antunes L (eds) Multi-agent-based simulation VI, vol 3891. Springer, Berlin/Heidelberg, pp 41–57
24. Holland JH (1992) Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, Cambridge, MA
25. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning, 1st edn. Addison-Wesley Professional, Boston
26. Camazine S (2001) Self-organization in biological systems. Princeton University Press, Princeton
27. Theraulaz G, Gautrais J, Camazine S, Deneubourg J-L (2003) The formation of spatial patterns in social insects: from simple behaviours to complex structures. Philos Trans R Soc London A: Math Phys Eng Sci 361(1807):1263–1282
28. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344 (2–3):243–278
29. Wilensky U (1998) NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston
30. Ordoukhanian E, Madni AM (2017) Introducing Resilience into Multi-UAV SoS. In: Presented at the 15th annual Conference on Systems Engineering Research (CSER), Redondo Beach
31. Madni AM, Sievers MW, Humann J, Ordoukhanian E, D'Ambrosio J, Sundaram P (2017) Model-based approach for engineering resilient system-of-systems: application to autonomous vehicle network. In: Presented at the 15th annual Conference on Systems Engineering Research (CSER), Redondo Beach