Emergence of Collective Behavior in Evolving Populations of Flying Agents

Lee Spector[1], Jon Klein[1,2], Chris Perry[1], and Mark Feinstein[1]

[1] School of Cognitive Science, Hampshire College
Amherst, MA 01002, USA
[2] Physical Resource Theory, Chalmers U. of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
{lspector, jklein, perry, mfeinstein}@hampshire.edu
http://hampshire.edu/lspector

Abstract. We demonstrate the emergence of collective behavior in two evolutionary computation systems, one an evolutionary extension of a classic (highly constrained) flocking algorithm and the other a relatively un-constrained system in which the behavior of agents is governed by evolved computer programs. We describe the systems in detail, document the emergence of collective behavior, and argue that these systems present new opportunities for the study of group dynamics in an evolutionary context.

1 Introduction

The evolution of group behavior is a central concern in evolutionary biology and behavioral ecology. Ethologists have articulated many costs and benefits of group living and have attempted to understand the ways in which these factors interact in the context of evolving populations. For example, they have considered the thermal advantages that warm-blooded animals accrue by being close together, the hydrodynamic advantages for fish swimming in schools, the risk of increased incidence of disease in crowds, the risk of cuckoldry by neighbors, and many advantages and risks of group foraging [4]. Attempts have been made to understand the evolution of group behavior as an optimization process operating on these factors, and to understand the circumstances in which the resulting optima are stable or unstable [6], [10]. Similar questions arise at a smaller scale and at an earlier phase of evolutionary history with respect to the evolution of symbiosis, multicellularity, and other forms of aggregation that were required to produce the first large, complex life forms [5], [1].

Artificial life technologies provide new tools for the investigation of these issues. One well-known, early example was the use of the Tierra system to study the evolution of a simple form of parasitism [7]. Game theoretic simulations, often based on the Prisoner's Dilemma, have provided ample data and insights, although usually at a level of abstraction far removed from the physical risks and opportunities presented by real environments (see, e.g., [2], about which we say a bit more below). Other investigators have attempted to study the evolution of collective behavior in populations of flying or swimming agents that are similar in some ways to those investigated here, with varying degrees of success [8], [13]. The latest wave of artificial life technology presents yet newer opportunities, however, as it is now possible to conduct much more elaborate simulations on modest hardware and in short time spans, to observe both evolution and behavior in real time in high-resolution 3D displays, and to interactively explore the ecology of evolving ecosystems.

In the present paper we describe two recent experiments in which the emergence of collective behavior was observed in evolving populations of flying agents. The first experiment used a system, called SwarmEvolve 1.0, that extends a classic flocking algorithm to allow for multiple species, goal orientation, and evolution of the constants in the hard-coded motion control equation. In this system we observed the emergence of a form of collective behavior in which species act similarly to multicellular organisms. The second experiment used a later and much-altered version of this system, called SwarmEvolve 2.0, in which the behavior of agents is controlled by evolved computer programs instead of a hard-coded motion control equation.[1] In this system we observed the emergence of altruistic food-sharing behaviors and investigated the link between this behavior and the stability of the environment.

Both SwarmEvolve 1.0 and SwarmEvolve 2.0 were developed within breve, a simulation package designed by Klein for realistic simulations of decentralized systems and artificial life in 3D worlds [3]. breve simulations are written by defining the behaviors and interactions of agents using a simple object-oriented programming language called steve. breve provides facilities for rigid body simulation, collision detection/response, and articulated body simulation. It simplifies the rapid construction of complex multi-agent simulations and includes a powerful OpenGL display engine that allows observers to manipulate the perspective in the 3D world and view the agents from any location and angle. The display engine also provides several "special effects" that can provide additional visual cues to observers, including shadows, reflections, lighting, semi-transparent bitmaps, lines connecting neighboring objects, texturing of objects and the ability to treat objects as light sources. More information about breve can be found in [3]. The breve system itself can be found on-line at http://www.spiderland.org/breve.

In the following sections we describe the two SwarmEvolve systems and the collective behavior phenomena that we observed within them. This is followed by some brief remarks about the potential for future investigations into the evolution of collective behavior using artificial life technology.

[1] A system that appears to be similar in some ways, though it is based on 2D cellular automata and the Santa Fe Institute Swarm system, is described at http://omicrongroup.org/evo/.

2 SwarmEvolve 1.0

One of the demonstration programs distributed with BREVE is SWARM, a simulation of flocking behavior modeled on the "boids" work of Craig W. Reynolds [9]. In the BREVE SWARM program the acceleration vector for each agent is determined at each time step via the following formulae:

$$\mathbf{V} = c_1\mathbf{V}_1 + c_2\mathbf{V}_2 + c_3\mathbf{V}_3 + c_4\mathbf{V}_4 + c_5\mathbf{V}_5$$

$$\mathbf{A} = m(\frac{\mathbf{V}}{|\mathbf{V}|})$$

The c_i are constants and the $\mathbf{V}_i$ are vectors determined from the state of the world (or in one case from the random number generator) and then normalized to length 1. $\mathbf{V}_1$ is a vector away from neighbors that are within a "crowding" radius, $\mathbf{V}_2$ is a vector toward the center of the world, $\mathbf{V}_3$ is the average of the agent's neighbors' velocity vectors, $\mathbf{V}_4$ is a vector toward the center of gravity of all agents, and $\mathbf{V}_5$ is a random vector. In the second formula we normalize the resulting velocity vector to length 1 (assuming its length is not zero) and set the agent's acceleration to the product of this result and m, a constant that determines the agent's maximum acceleration. The system also models a floor and hard-coded "land" and "take off" behaviors, but these are peripheral to the focus of this paper. By using different values for the c_i and m constants (along with the "crowding" distance, the number of agents, and other parameters) one can obtain a range of different flocking behaviors; many researchers have explored the space of these behaviors since Reynolds's pioneering work [9].

SwarmEvolve 1.0 enhances the basic BREVE SWARM system in several ways. First, we created three distinct species[2] of agents, each designated by a different color. As part of this enhancement we added a new term, $c_6\mathbf{V}_6$, to the motion formula, where $\mathbf{V}_6$ is a vector away from neighbors *of other species* that are within a "crowding" radius. Goal-orientation was introduced by adding a number of randomly moving "energy" sources to the environment and imposing energy dynamics. As part of this enhancement we added one more new term, $c_7\mathbf{V}_7$, to the motion formula, where $\mathbf{V}_7$ is a vector toward the nearest energy source. Each time an agent collides with an energy source it receives an energy boost (up to a maximum), while each of the following bears an energy cost:

– Survival for a simulation time step (a small "cost of living").
– Collision with another agent.
– Being in a neighborhood (bounded by a pre-set radius) in which representatives of the agent's species are outnumbered by representatives of other species.
– Giving birth (see below).

[2] "Species" here are simply imposed, hard-coded distinctions between groups of agents, implemented by filling "species" slots in the agent data structures with integers ranging from 0 to 2. This bears only superficial resemblance to biological notions of "species."

The numerical values for the energy costs and other parameters can be adjusted arbitrarily and the effects of these adjustments can be observed visually and/or via statistics printed to the log file; values typical of those that we used can be found in the source code for SwarmEvolve 1.0.[3]

As a final enhancement we leveraged the energy dynamics to provide a fitness function and used a genetic encoding of the control constants to allow for evolution. Each individual has its own set of c_i constants; this set of constants controls the agent's behavior (via the enhanced motion formula) and also serves as the agent's genotype. When an agent's energy falls to zero the agent "dies" and is "reborn" (in the same location) by receiving a new genotype and an infusion of energy. The genotype is taken, with possible mutation (small perturbation of each constant) from the "best" current individual of the agent's species (which may be at a distant location).[4] We define "best" here as the *product* of energy and age (in simulation time steps). The genotype of the "dead" agent is lost, and the agent that provided the genotype for the new agent pays a small energy penalty for giving birth. Note that reproduction is asexual in this system (although it may be sexual in SwarmEvolve 2.0).

The visualization system presents a 3D view (automatically scaled and targeted) of the geometry of the world and all of the agents in real time. Commonly available hardware is sufficient for fluid action and animation. Each agent is a cone with a pentagonal base and a hue determined by the agent's species (red, blue, or purple). The color of an agent is dimmed in inverse proportion to its energy — agents with nearly maximal energy glow brightly while those with nearly zero energy are almost black. "Rebirth" events are visible as agents flash from black to bright colors.[5] Agent cones are oriented to point in the direction of their velocity vectors. This often produces an appearance akin to swimming or to "swooping" birds, particularly when agents are moving quickly. Energy sources are flat, bright yellow pentagonal disks that hover at a fixed distance above the floor and occasionally glide to new, random positions within a fixed distance from the center of the world. An automatic camera control algorithm adjusts camera zoom and targeting continuously in an attempt to keep most of the action in view.

Figure 1 shows a snapshot of a typical view of the SwarmEvolve world. An animation showing a typical action sequence can be found on-line.[6]

SwarmEvolve 1.0 is simple in many respects but it nonetheless exhibits rich evolutionary behavior. One can often observe the species adopting different strategies; for example, one species often evolves to be better at tracking quickly moving energy sources, while another evolves to be better at capturing static en-

[3] http://hampshire.edu/lspector/swarmevolve-1.0.tz

[4] The choice to have death and rebirth happen in the same location facilitated, as an unanticipated side effect, the evolution of the form of collective behavior described below. In SwarmEvolve 2.0, among many other changes, births occur near parents.

[5] Birth energies are typically chosen to be random numbers in the vicinity of half of the maximum.

[6] http://hampshire.edu/lspector/swarmevolve-ex1.mov

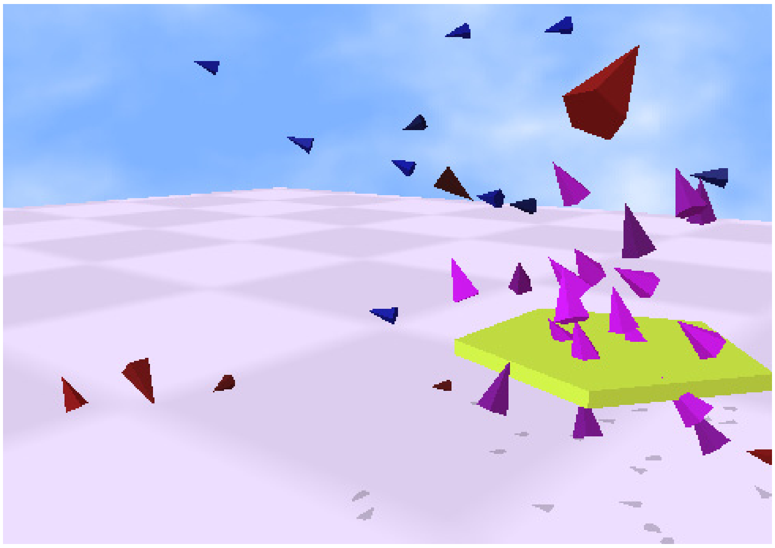

Fig. 1. A view of SwarmEvolve 1.0 (which is in color but will print black and white in the proceedings). The agents in control of the pentagonal energy source are of the purple species, those in the distance in the upper center of the image are blue, and a few strays (including those on the left of the image) are red. All agents are the same size, so relative size on screen indicates distance from the camera.

ergy sources from other species. An animation demonstrating evolved strategies such as these can be found on-line.[7]

3 Emergence of Collective Behavior in SwarmEvolve 1.0

Many SwarmEvolve runs produce at least some species that tend to form static clouds around energy sources. In such a species, a small number of individuals will typically hover within the energy source, feeding continuously, while all of the other individuals will hover in a spherical area surrounding the energy source, maintaining approximately equal distances between themselves and their neighbors. Figure 2 shows a snapshot of such a situation, as does the animation at http://hampshire.edu/lspector/swarmevolve-ex2.mov; note the behavior of the purple agents.

We initially found this behavior puzzling as the individuals that are not actually feeding quickly die. On first glance this does not appear to be adaptive behavior, and yet this behavior emerges frequently and appears to be relatively stable. Upon reflection, however, it was clear that we were actually observing the emergence of a higher level of organization.

When an agent dies it is reborn, in place, with a (possibly mutated) version of the genotype of the "best" current individual of the agent's species, where

[7] http://hampshire.edu/lspector/swarmevolve-ex2.mov

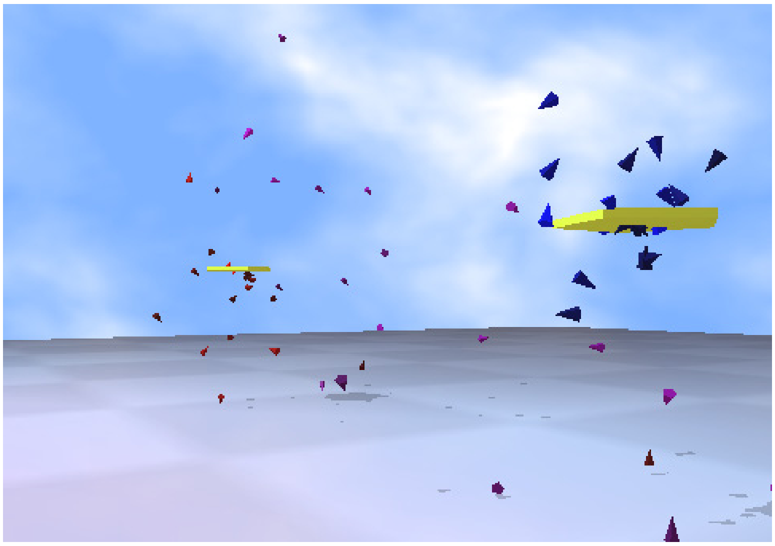

Fig. 2. A view of SwarmEvolve 1.0 in which a cloud of agents (the blue species) is hovering around the energy source on the right. Only the central agents are feeding; the others are continually dying and being reborn. As described in the text this can be viewed as a form of emergent collective organization or multicellularity. In this image the agents controlling the energy source on the left are red and most of those between the energy sources and on the floor are purple.

quality is determined from the product of age and energy. This means that the new children that replace the dying individuals on the periphery of the cloud will be near-clones of the feeding individuals within the energy source. Since the cloud generally serves to repel members of other species, the formation of a cloud is a good strategy for keeping control of the energy source. In addition, by remaining sufficiently spread out, the species limits the possibility of collisions between its members (which have energy costs). The high level of genetic redundancy in the cloud is also adaptive insofar as it increases the chances that the genotype will survive after a disruption (which will occur, for example, when the energy source moves).

The entire feeding cloud can therefore be thought of as a genetically coupled collective, or even as a multicellular organism in which the peripheral agents act as defensive organs and the central agents act as digestive and reproductive organs.

4 SwarmEvolve 2.0

Although SwarmEvolve 2.0 was derived from SwarmEvolve 1.0 and is superficially similar in appearance, it is really a fundamentally different system.

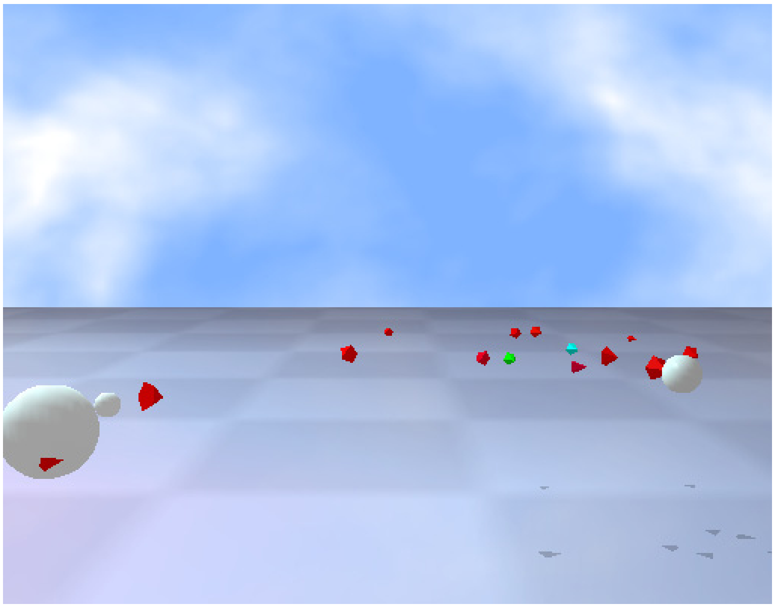

Fig. 3. A view of SwarmEvolve 2.0 in which energy sources shrink as they are consumed and agents are "fatter" when they have more energy.

The energy sources in SwarmEvolve 2.0 are spheres that are depleted (and shrink) when eaten; they re-grow their energy over time, and their signals (sensed by agents) depend on their energy content and decay over distance according to an inverse square law. Births occur near mothers and dead agents leave corpses that fall to the ground and decompose. A form of energy conservation is maintained, with energy entering the system only through the growth of the energy sources. All agent actions are either energy neutral or energy consuming, and the initial energy allotment of a child is taken from the mother. Agents get "fatter" (the sizes of their bases increase) when they have more energy, although their lengths remain constant so that length still provides the appropriate cues for relative distance judgement in the visual display. A graphical user interface has also been added to facilitate the experimental manipulation of system parameters and monitoring of system behavior.

The most significant change, however, was the elimination of hard-coded species distinctions and the elimination of the hard-coded motion control formula (within which, in SwarmEvolve 1.0, only the constants were subject to variation and evolution). In SwarmEvolve 2.0 each agent contains a computer program that is executed at each time step. This program produces two values that control the activity of the agent:

1. a vector that determines the agent's acceleration,
2. a floating-point number that determines the agent's color.

Agent programs are expressed in Push, a programming language designed by Spector to support the evolution of programs that manipulate multiple data types, including code; the explicit manipulation of code supports the evolution of modules and control structures, while also simplifying the evolution of agents that produce their own offspring rather than relying on the automatic application of hand-coded crossover and mutation operators [11], [12].

Table 1. Push instructions available for use in SwarmEvolve 2.0 agent programs

Instruction(s)	Description
DUP, POP, SWAP, REP, =, NOOP, PULL, PULLDUP, CONVERT, CAR, CDR, QUOTE, ATOM, NULL, NTH, +, $*$, /, >, <, NOT, AND, NAND OR, NOR, DO*, IF	Standard Push instructions (See [11])
VectorX, VectorY, VectorZ, VPlus, VMinus, VTimes, VDivide, VectorLength, Make-Vector	Vector access, construction, and manipulation
RandI, RandF, RandV, RandC	Random number, vector, and code generators
SetServoSetpoint, SetServoGain, Servo	Servo-based persistent memory
Mutate, Crossover	Stochastic list manipulation (parameters from stacks)
Spawn	Produce a child with code from code stack
ToFood	Vector to energy source
FoodIntensity	Energy of energy source
MyAge, MyEnergy, MyHue, MyVelocity, MyLocation, MyProgram	Information about self
ToFriend, FriendAge, FriendEnergy, FriendHue, FriendVelocity, FriendLocation, FriendProgram	Information about closest agent of similar hue
ToOther, OtherAge, OtherEnergy, OtherHue, OtherVelocity, OtherLocation, OtherProgram	Information about closest agent of non-similar hue
FeedFriend, FeedOther	Transfer energy to closest agent of indicated category

The Push instructions available for use in agent programs are shown in Table 1. In addition to the standard Push instructions, operating on integers, floating point numbers, Boolean values, and code expressions, instructions were added for the manipulation of vectors and for SwarmEvolve-specific sensors and actions. Note that two sets of instructions are provided for getting information about other agents in the world, the "friend" instructions and the "other" instructions. Each "friend" instruction operates on the closest agent having a color similar to the acting agent (currently defined as having a hue within 0.1 in a hue scale that ranges from 0.0 to 1.0). Each "other" instruction operates on the closest agent having a color that is *not* similar to the acting agent.[8] In some cases, in particular when an agent sets its color once and never changes it, the "friend" instructions will be likely to operate on relatives, since presumably these relatives would set their colors similarly. But since agents can change their colors dynamically each time-step, a "friend" is not necessarily a relative and a relative is not necessarily a "friend." The term "friend" here should be taken with a grain of salt; the friend/other distinction provides a way for agents to distinguish among each other based on color, but they may use this capability in a variety of ways.

SwarmEvolve 2.0 is an "autoconstructive evolution" system, in which agents are responsible for producing their own offspring and arbitrary reproductive mechanisms may evolve [11]. Whenever an agent attempts to produce a child (by executing the Spawn instruction), the top of its code stack is examined. If the expression is empty (which happens rarely once the system has been running for some time) then a newly generated, random program is used for the child. If the expression is not empty then it is used as the child's program, after a possible mutation. The probability of mutation is also determined by the parent. A random number is chosen from a uniform distribution from zero to the absolute value of the number on top of the Integer stack; if the chosen number is zero then a mutation is performed. The mutation operation is similar to that used in traditional genetic programming: a random sub-expression is replaced with a newly generated random expression. Note that the program access instructions provide the means for agents to produce their children asexually or sexually, potentially using code from many "mates."

At the beginning of a SwarmEvolve 2.0 run most of the agents, which will have been generated randomly, will not have programs that cause them to seek food and produce offspring; they will therefore die rather quickly and the population will plummet. Whenever the population drops below a user-defined threshold the system injects new random agents into the world. With the parameters used here, however, it usually takes only a few hundred time-steps before "reproductive competence" is achieved — at this point the population is self-sustaining as there are a large number of agents capable of reproducing.

SwarmEvolve 2.0 is a complex program with many parameters, not all of which can be addressed in the scope of this short paper. However, the source code for the system (including the parameters used in the experiments described below) is available on-line.[9] Figure 3 shows a typical scene from SwarmEvolve 2.0; an animation of a typical action sequence can be found on-line.[10]

[8] If there are no other agents meeting the relevant criterion then each of these instructions operates on the acting agent itself.

[9] http://hampshire.edu/lspector/swarmevolve-2.0.tz

[10] http://hampshire.edu/lspector/swarmevolve2-ex1.mov

5 Emergence of Collective Behavior in SwarmEvolve 2.0

The last two instructions listed in Table 1, FeedFriend and FeedOther, provide a means for agents to transfer energy to one another (to share food). Each of these instructions transfers a small increment of energy (0.01 out of a possible total of 1.0), but only under certain conditions which we varied experimentally (see below). Ordinarily, the use of these instructions would seem to be maladaptive, as they decrease the energy of the acting agents. The use of a Feed instruction thereby makes the feeding agent both more vulnerable and less likely to produce children.

Might there nonetheless be some circumstances in which it is adaptive for agents to feed one another? We set out to investigate this question by conducting runs of SwarmEvolve 2.0 and monitoring the proportion of agents that feed or attempt to feed other agents.[11] Because the Feed instructions will occasionally occur in randomly generated code and in mutations, we expect every run to produce some number of calls to these instructions. We expect, however, that the proportion of food sharing agents, when averaged over a large number of runs, will reflect the extent to which food sharing is adaptive.

We hypothesized, for example, that dynamic, unstable environments might provide a breeding ground for altruistic feeding behavior. We reasoned as follows, from the perspective of a hypothetical agent in the system: "If the world is stable, and everyone who's smart enough to find food can reliably get it, then I should get it when I can and keep it to myself. If the world is unstable, however, so that I'll sometimes miss the food despite my best efforts, then it'd be better for me if everyone shared. Food from others would help to buffer the effects of chance events, and I'd be willing to share food when I have it in order to secure this kind of insurance." Of course one shouldn't put too much faith in such "just so stories," but they can sometimes be a guide for intuitions. In the present case they led us to conduct a simple experiment in which we varied the stability of the energy sources and the sharing conditions in SwarmEvolve 2.0 and measured the proportion of food-sharing agents that resulted.

We conducted a total of 1,625 runs under a variety of stability and sharing conditions. We used values of the "stability" parameter ranging from 20 (unstable) to 2,000 (highly stable). The stability parameter governs the frequency with which energy sources begin to drift to new, random locations; the probability that a particular energy source will begin to drift to a new location in any particular time step is $\frac{1}{stability}$. We collected data on four different sharing conditions. In all of the conditions the potential recipient is the closest agent of similar or dissimilar color, depending on whether the agent is executing the FeedFriend or FeedOther instruction respectively. In all cases the feeding is conditional on the recipient having less energy than the provider. In "waste" sharing the energy is all lost in the transfer, and the recipient receives nothing; we included this

[11] For the analyses presented below we did not distinguish between FeedFriend and FeedOther executions; we explored the distinction briefly but there were no obvious patterns in the data.

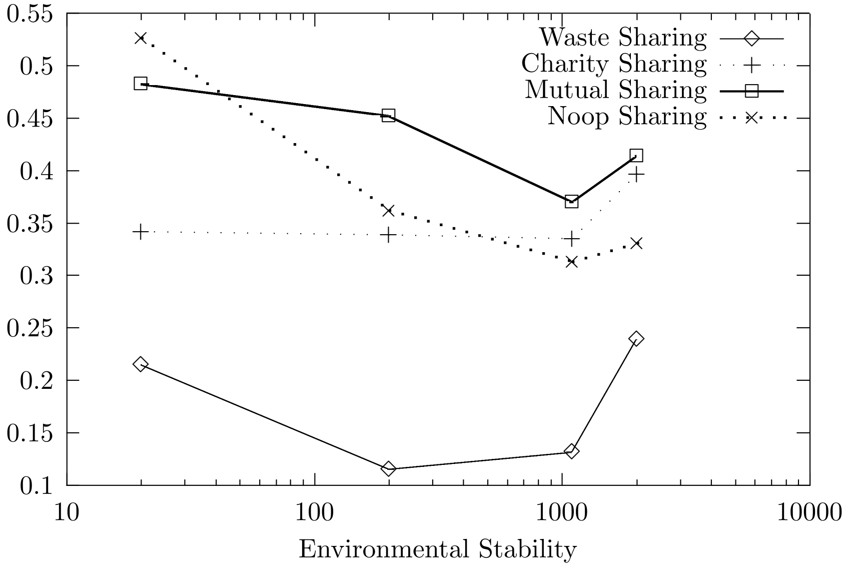

Fig. 4. Proportion of agents that share food (on the y axis) graphed vs. environmental (energy source) stability (on the x axis) for four sharing conditions (see text).

sharing condition as a control. In "charity" sharing the recipient receives all of the energy, regardless of whether or not the recipient itself shares energy. In "mutual" sharing the recipient receives all of the energy, but only if it has itself shared energy at least once in its life. Finally, in "noop" sharing no energy is transferred or lost; this is another control.

We collected data only from runs with at least 5,300 consecutive time-steps of reproductive competence — there were 936 runs meeting this condition. For qualifying runs we then collected data over the last 5,000 of time-steps, divided into 100-time-step "epochs." At each epoch boundary we took a census, recording the proportion of living agents that have attempted to share energy with another agent on at least one occasion.

Our results are graphed in Figure 4. Our hypothesis that dynamic, unstable environments might provide a breeding ground for altruistic feeding behavior was only partially confirmed; indeed, the most stable environments also appear to be conducive to food sharing. To the extent that the hypothesis is confirmed it is interesting to note that a similar effect, involving the preference for cooperation in unpredictable environments, has been observed in a radically different, game-theoretic context [2].

The "waste" sharing control produced less food sharing than all of the other sharing conditions; this is what one would expect, as "waste" sharing has costs but no possible benefits. The "noop" sharing control, on the other hand, which has no costs and no benefits, produced slightly more sharing than all other conditions at low stability. Note, however, that both "charity" sharing and "mutual" sharing, which have both costs and potential benefits, produced more sharing than both of the controls at several stability settings. There is substantial variance in the data, and the statistical significance of some of the differences visible in the graph is questionable. In any event we can say that the amount of sharing in the "charity" and "mutual" conditions was, under several stability settings, either greater than or at least not significantly less than the amount of sharing in the "noop" control. This by itself is evidence that altruistic feeding behavior is adaptive in the environment under study. Many of the other differences in the data are clearly significant, and the trends indicate that collective feeding behaviors do arise in some circumstances and not in others. These simulations provide a rich framework for investigating the relations between collective behavior and evolution, which we have only begun to explore.

6 Conclusions and Future Work

The emergence of collective behavior is an intriguing and at times counterintuitive phenomenon, an understanding of which will have significant impacts on the study of living systems at all levels, from symbiotic microbes to human societies. The work presented in this paper demonstrates that new artificial life technologies provide new tools for the synthetic investigation of these phenomena, complementing the well-established analytic methods of evolutionary biology and behavioral ecology.

In particular we demonstrated the emergence of a simple form of multicellular organization in evolving populations of agents based on a traditional flocking algorithm. We also demonstrated the emergence of altruistic feeding behavior in a system that is considerably less constrained, as the agents are controlled by evolved computer programs. We believe that this latter system provides significant new avenues of study by allowing for agents of arbitrary complexity to evolve within complex, dynamic worlds.

Our own plans for this work in the near future include a systematic exploration of the effects of various parameter changes on the emergence of collective behavior. We are making the source code for SwarmEvolve 1.0 and 2.0 freely available in the hopes that others will also contribute to this process; see
http://hampshire.edu/lspector/swarmevolve-1.0.tz and
http://hampshire.edu/lspector/swarmevolve-2.0.tz.

Acknowledgments. Raymond Coppinger, Rebecca Neimark, Wallace Feurzeig, Oliver Selfridge, and three anonymous reviewers provided comments that helped to improve this work. This effort was supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30502-00-2-0611, and by NSF grant EIA-0216344.

References

1. Bonner, J.T.: The evolution of complexity by means of natural selection. Princeton University Press, Princeton, NJ (1988)
2. Eriksson, A., Lindgren, K.: Cooperation in an Unpredictable Environment. Proc. Eighth Intl. Conf. on Artificial Life. The MIT Press, Cambridge, MA (2002) 394–399.
3. Klein, J.: breve: a 3D Environment for the Simulation of Decentralized Systems and Artificial Life. Proc. Eighth Intl. Conf. on Artificial Life. The MIT Press, Cambridge, MA (2002) 329–334. http://www.spiderland.org/breve/breve-klein-alife2002.pdf
4. Krebs, J.R., Davies, N.B.: An Introduction to Behavioural Ecology. 3rd edn. Blackwell Scientific Publications LTD, Oxford London Edinburgh Boston Melbourne Paris Berlin Vienna (1981)
5. Maynard Smith, J., Szathmáry, E.: The origins of life. Oxford University Press (1999)
6. Pulliam, H.R., Caraci, T.: Living in groups: is there an optimal group size? In J.R. Krebs and N.B. Davies (eds), Behavioral Ecology: An Evolutionary Approach, 2nd edn. Blackwell Scientific Publications LTD, Oxford London Edinburgh Boston Melbourne Paris Berlin Vienna (1984) 122–147
7. Ray, T.S.: Is it Alive or is it GA. Proc. Fourth Intl. Conf. on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991) 527–534
8. Reynolds, C.W.: An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion. In From Animals to Animats 2: Proc. Second Intl. Conf. on Simulation of Adaptive Behavior. The MIT Press, Cambridge, MA (1993) 384–392
9. Reynolds, C.W.: Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics **24**:4 (1987) 25–34
10. Silby, R.M.: Optimal group size is unstable. Anim. Behav. **31** (1983) 947–948
11. Spector, L., Robinson, A.: Genetic Programming and Autoconstructive Evolution with the Push Programming Language. Genetic Programming and Evolvable Machines **3**:1 (2002) 7–40
12. Spector, L.: Adaptive Populations of Endogenously Diversifying Pushpop Organisms are Reliably Diverse. Proc. Eighth Intl. Conf. on Artificial Life. The MIT Press, Cambridge, MA (2002) 142–145
13. Zaera, N., Cliff, D., Bruten, J.: (Not) Evolving Collective Behaviours in Synthetic Fish. From Animals to Animats 4: Proc. Second Intl. Conf. on Simulation of Adaptive Behavior. The MIT Press, Cambridge, MA (1996)