Proceedings of the 10th European Conference on
Artificial Life, Budapest, Hungary, September 2009.

Evolving Virtual Fireflies

David B. Knoester and Philip K. McKinley

Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
{dk, mckinley}@cse.msu.edu

Abstract. In this paper, we present a study in the evolution of cooperative behavior, specifically synchronization, through digital evolution and multilevel selection. In digital evolution, a population of self-replicating computer programs exists in a user-defined computational environment and is subject to instruction-level mutations and natural selection. Multilevel selection links the survival of the individual to the survival of its group, thus encouraging cooperation. Previous approaches to *designing* synchronization algorithms have taken inspiration from nature: In the well-known firefly model, the only form of communication between agents is in the form of "flash" messages among neighbors. Here we demonstrate that populations of digital organisms, provided with a similar mechanism and minimal information about their environment, are capable of *evolving* algorithms for synchronization, and that the evolved behaviors are robust to message loss. Moreover, analysis of the dominant genome reveals that the evolved solution utilizes an adaptive frequency strategy strikingly similar to that observed in fireflies.

Keywords: evolutionary computation, digital evolution, synchronization, self-organization, cooperative behavior, distributed algorithm.

1 Introduction

The natural world is replete with organisms that exhibit cooperative behaviors of varying complexity. Some of these cooperative behaviors exhibit *synchrony*. For example, honey bees synchronize their activity cycles [1], fiddler crabs synchronously wave their oversized claw [2], and ants synchronize their alarm drumming [3]. One of the more striking examples of synchrony in the natural world is the coordinated flashing of male fireflies. In some parts of Southeast Asia, these fireflies synchronize their flashes to a common period and phase over a distance of many miles [4]. Researchers have developed several mathematical models of this behavior, among them the pulse-coupled oscillator model of Mirollo and Strogatz [5] and the adaptive frequency model of Ermentrout [6]. Such models enable the design of *biomimetic* synchronization algorithms. For example, Babaoglu et al. [7] leveraged the Ermentrout model to develop a heartbeat synchronization algorithm for large overlay networks, facilitating coordination of collective tasks among network nodes.

In the study reported here, we used AVIDA [8], a digital evolution platform closely associated with artificial life, to explore the evolution of synchronization behavior. In

AVIDA, a population of self-replicating computer programs (digital organisms) exists in a user-defined environment and is subject to mutation and natural selection. We extended AVIDA with a small set of instructions that enable digital organisms to transmit and receive virtual "flash" messages. We also defined multilevel selection criteria that reward groups of digital organisms for exhibiting synchronization behavior. In experiments, the AVIDA populations evolved the ability to synchronize very quickly from arbitrary initial states. Analysis of the dominant genome revealed that the solution utilizes an adaptive frequency strategy remarkably similar to that of the Ermentrout model. While other studies have investigated many aspects of the evolution of cooperative behavior [9–11], including synchrony [12], the main contribution of this work is to demonstrate the *de novo* evolution of a cooperative behavior for synchronization. Moreover, our experiments show that this synchronization behavior evolves even in the presence of significant loss rates of virtual flashes, suggesting that an adaptive frequency mechanism is an important element in resiliency to environmental interference.

2 Research Platform

Digital evolution [13] is a form of evolutionary computation originally developed to study evolution in biology. AVIDA [8], a platform for digital evolution, is well-suited for studies of cooperative behavior, and has previously been used in artificial life studies on the evolution of cooperative communication behaviors [14] and adaptive sleep response [15].

Figure 1(a) depicts an AVIDA population and the structure of an individual organism. Each digital organism comprises a circular list of instructions (its *genome*) and a virtual CPU, and exists in a common virtual environment. The virtual CPU contains three general-purpose registers (AX, BX, CX), two stacks, and a number of *heads* (pointers to instructions in the genome), which can be manipulated for execution-flow control. Within their environment, organisms execute the instructions in their genomes, and the particular instructions that are executed determine the organism's behavior (its *phenotype*). Instructions within an organism's genome are similar in appearance and functionality to traditional assembly language instructions. These instructions enable an organism to perform simple mathematical operations, such as addition, multiplication, and bit-shifts; to manipulate the position of heads within their genome; to sense and change properties of the environment; and to communicate with neighboring organisms. Instructions within AVIDA can also have different costs in terms of virtual CPU cycles. For example, a simple addition may cost only one cycle, while broadcasting a message may cost 20 cycles. New instructions implemented for this study are summarized in Table 1. Of particular relevance is the flash instruction, which broadcasts a message to each of the calling organism's neighbors within the virtual environment. Organisms can retrieve information about any received flashes via the if-recvd-flash, flash-info, and flash-info-b instructions.

Many approaches to the evolution of cooperation in non-biological systems involve *multilevel selection*, where selection not only acts on individuals, but also on the groups to which the individuals belong. AVIDA provides a framework for multilevel selection called CompeteDemes, which enables the periodic replication and competition among *demes*. In AVIDA, a deme is an isolated subpopulation of organisms. In Fig-

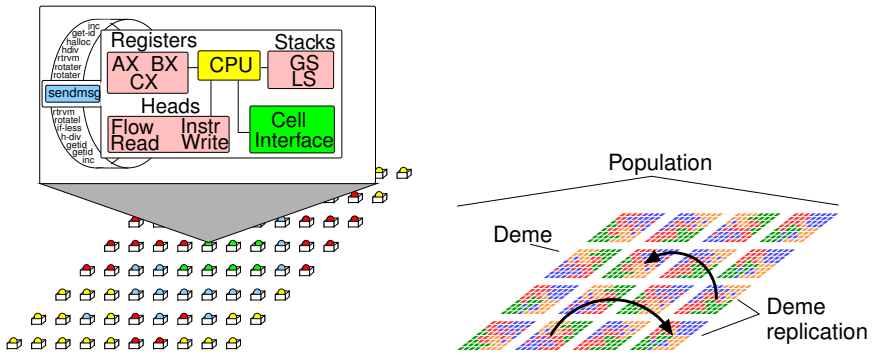


Fig. 1. Elements of the AVIDA platform: (a) an AVIDA population containing multiple genomes (bottom) and the structure of an individual organism (top); (b) depiction of an AVIDA population divided into 16 demes. When a deme replicates, it replaces a randomly selected target deme.

ure 1(b), we see a population divided into 16 demes. During the execution of an AVIDA experiment, the CompeteDemes framework periodically calculates the fitness of each deme via a user-defined fitness function. This fitness function takes as input a single deme, and produces the fitness of that deme (a floating-point number) as output. Using the resulting array of fitness values, the CompeteDemes framework then performs fitness-proportional selection, preferentially replicating those demes with higher fitness. For example, one may define a fitness function based on completing a cooperative task. Over time, the CompeteDemes framework will then preferentially replicate those demes that are more successful than others.

Table 1. New AVIDA virtual CPU instructions implemented for this study. All instructions are equally likely to be selected as targets for mutation.

Instruction	Description
flash	Broadcasts a "flash" message to caller's neighbors, with a configurable loss rate.
if-recvd-flash	If the caller has received a flash from any of its neighbors, execute the subsequent instruction. Otherwise, skip the subsequent instruction.
flash-info	If the caller has ever received a flash, set BX to 1 and CX to the number of cycles since that flash was received. Otherwise, set BX and CX to 0.
flash-info-b	If the caller has ever received a flash, set BX to 1; do not modify CX.
hard-reset	Reset the state of the virtual CPU to the organism's "birth" state. All registers are zeroed out and heads are reset, including the flash timer and cycle counter.
get-cycles	Set BX to the number of virtual CPU cycles since either the organism was born or the last time hard-reset was called, whichever is most recent.

To encourage the evolution of cooperation, we also employed *digital germlines* [16], a framework that provides a single common genetic ancestry for all organisms within a deme. In AVIDA, the germline is a genome attached to a deme, rather than to an individual. Although individuals within a deme can self-replicate, mutations occur only along the germline, and then only during deme replication. When a deme replicates (the

arrows in Figure 1(b)), the germline for the source deme is copied (subject to mutation) to the target deme, and an organism constructed from that germline is inserted into the target deme. The use of a digital germline has the side-effect of homogenizing the inhabitants of each deme, a technique that has been effective in evolutionary robotics [10]. Moreover, in an earlier study, we observed that using a digital germline was necessary to evolve organisms that cooperated to construct communication networks [16].

3 Experiments and Results

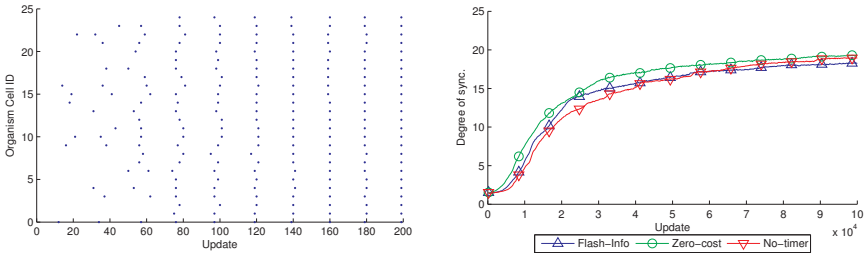
In this study we tested several different different combinations of instruction sets and environmental configurations for their ability to evolve synchronization behavior. Due to space limitations, here we only describe the configuration that was most effective; additional details can be found in an accompanying technical report [17]. We configured AVIDA with 400 5×5 toroidal demes, the default mutation rates (approximately 1 mutation per genome replication), and we employed a CompeteDemes period of 200 *updates*, where an update is the unit of virtual time in AVIDA corresponding to five virtual CPU cycles per organism. We initialized each deme to be in a state of desynchronization by starting from a single organism, and inserting exactly one new organism with a 50% probability each update. We configured the CompeteDemes framework to compete all 400 demes with each other every 200 updates, according to the following fitness function:

$$F = \begin{cases} 1 + U & \text{if } U < S \\ 1 + U + (flash_{max} - flash_{mean})^2 & \text{if } U \geq S \end{cases} \quad (1)$$

where F is the fitness of the deme, U is the number of unique organisms that issued a flash, S is the number of organisms in the deme (always 25 in this study), $flash_{max}$ is the maximum number of flashes during any single update, and $flash_{mean}$ is the mean number of flashes per update. Both $flash_{max}$, $flash_{mean}$, and U were calculated from the final 50 updates of each CompeteDemes period to allow organisms a time (the first 150 updates) during which they may issue flash instructions without adversely affecting the deme's fitness. This fitness function thus rewards demes whose constituent organisms each execute the flash instruction, and then further rewards demes for increasing the difference between the maximum and mean number of flashes per update. This definition of synchronization is similar to that in [7], where it is not required that all flashes occur at *exactly* the same point in time, but rather within a small window.

Figure 2(a) is a detail of the behavior of a single deme containing 25 copies of the *dominant* (that is, most common) evolved genome from the end of one of the best-performing of 30 AVIDA trials. Each point in this plot represents the execution of the flash instruction by a particular organism. Here we see runtime synchronization, where the 25 individuals within the deme have evolved to synchronize their flashes at an identical phase and frequency within 200 updates. Of note here is that evolution has solved *two* different problems: the period of successive flashes, and the behavior that brings them into synchronization.

To better understand the factors that influenced the evolution of synchronization, we conducted two additional treatments of the synchronization experiment. The *flash-info*



(a) (b)

Fig. 2. Experimental results showing (a) detail of evolved synchronization behavior; organisms initially flash asynchronously, but gradually synchronize with each other, and (b) degree of synchronization measured as the difference between the maximum and mean number of flashes for three treatments.

treatment, which evolved the behavior seen in Figure 2(a), used a 20-cycle cost for execution of the flash instruction, and was allowed to use the flash-info instruction. The *zero-cost* treatment modifies the flash-info treatment by reducing the cost of executing the flash instruction to one cycle, making the flash instruction no more expensive to execute than any other instruction in the genome. Finally, the *no-timer* treatment modifies the flash-info treatment by replacing the flash-info instruction with flash-info-b, preventing organisms from accessing timing information related to the reception of flash messages.

Figure 2(b) plots the difference between the maximum and mean number of flashes, calculated as part of fitness, averaged over each deme in 30 AVIDA trials, for each of these three different treatments. Here we see that after 100,000 updates, the mean difference between maximum and mean flash counts approaches 20 in all three treatments. In other words, after 500 generations (100,000 updates / 200 updates per CompeteDemes period) 80% of the organisms within the average deme in each treatment synchronized with each other, and we see that this behavior does not depend on instruction cost or knowledge of the exact timing of message reception.

Finally, we examined the effect of a flash loss rate on the evolution of synchronization. Specifically, we varied loss rate from 0% to 80%, with each treatment using a unit-cost flash instruction. We define a flash "loss" as a flash message that is lost in sending, that is, it is not received by any of its neighbors. Figure 3 plots the number of coincident flashes versus time averaged over each deme, in 10 trials for each of the different flash loss rates from 0% to 80%. Here we see that higher loss rates inhibit the evolution of synchronization behavior, though we note that evolution was still able to discover solutions comparable to the 0% loss rate case at loss rates up to 20%.

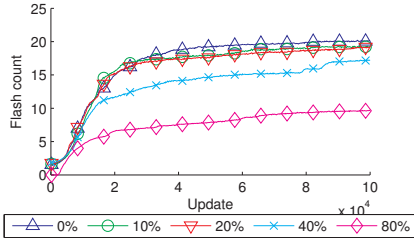
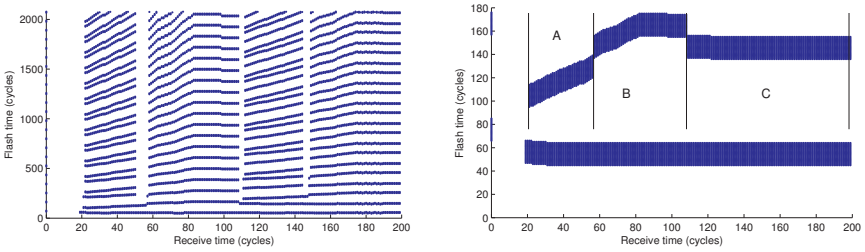


Fig. 3. Evolution of synchronization for different flash loss rates.

4 Genome Analysis

Let is next analyze the genome responsible for the behavior shown in Figure 2(a), a dominant drawn from a single AVIDA trial. This treatment used a 20-cycle cost for the flash instruction, and included the flash-info instruction, which provided the organism with information about the last flash received. The AVIDA TestCPU enables the analysis of a single organism in an isolated environment. To analyze synchronization behavior, we extended the AVIDA TestCPU to support the artificial delivery of a flash message to the organism under test. We then traced the organism's execution flow that resulted from receiving a flash at different times, and monitored its response in terms of its own execution of the flash execution.

Figure 4(a) plots the response of the dominant to receiving a flash at various times during its life. At each point t along the horizontal axis, we initiated a new test of the dominant, and artificially sent a single flash message to the organism once it had executed t virtual CPU cycles. We then monitored the response of the organism from each test for 2000 cycles, and plotted a point for each cycle spent executing the flash instruction. For example, in Figure 4(a) at time 100, we see a series of horizontal bands every 110 cycles. This indicates that if the organism receives a single flash after it has executed 100 instructions, its resulting behavior will be to flash every 110 cycles. The gaps from time 1 to 19 and near times 50, 110, and 150 are artifacts of the organism executing the hard-reset instruction.



(a) (b)

Fig. 4. Genome analysis results showing (a) synchronization response of the dominant genome to receiving flash messages at different times, and (b) detailed synchronization response of this genome to receiving flash messages at different times. Regions A and B are frequency-increasing and frequency-decreasing responses, respectively, while region C is a steady-state response.

Figure 4(b) zooms in on the first 180 cycles of the organism's response to receiving flash messages at different times. Here we see evidence of the strategy employed for synchronization. First, the horizontal band extending from time 19 to 200 indicates that, regardless of flash reception, organisms will issue a flash that will complete by their 66th virtual CPU cycle. Region C in Figure 4(b), extending from time 108 to 200, shows that the organism has the same response to receiving a flash at any point in this region, indicating that the organism is not making any phase or frequency adjustments to its flash execution. Region A, on the other hand, shows an earlier execution of flash than region C, indicating that the reception of a message in this region causes

a frequency-advance of the receiver. Finally, region B shows a slower response to flash messages than region C, indicating a region of frequency-delay. This combination of steady-state and adaptive frequency operations is strikingly similar to the models of biological synchronization from Mirollo, Strogatz, and Ermentrout [5, 6].

Figure 5 depicts the instructions present in the dominant genome, and shows the primary instructions that are responsible for the behavior in Figure 4(b). Upon birth, the organism executes the first 64 instructions in order, unless reception of a flash has triggered a reset, as mentioned earlier. Otherwise, periodic execution of the flash instruction begins at cycle 65. When the organism receives a flash, however, its execution flow dramatically changes. Specifically, it uses a combination of the hard-reset, get-cycles, flash-info, and jmp-head instructions to modify the position of the virtual CPU's instruction pointer. Depending on where this instruction pointer is moved to, the next execution of flash will either be earlier, later, or unchanged relative to the organism's natural frequency (110 cycles, as shown by Figure 4(a)). The genome shown in Figure 5 is annotated with the regions corresponding to these different behaviors. The genome also makes extensive use of conditional logic, via the if-recvd-flash instruction, to retrieve the numeric position of the instruction pointer within the genome. Finally, we note the large number of instructions (75) within this genome that have no immediately discernible purpose (instructions not shown). These instructions are most likely neutral mutations, or mutations that do not adversely affect fitness, though they may have had an important role earlier in the evolutionary process.

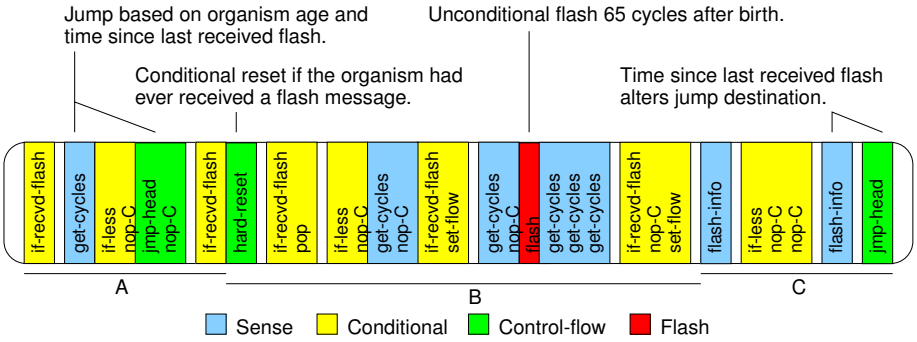


Fig. 5. Depiction of the dominant genome for synchronization. Labels (A, B, C) refer to destinations of the jmp-head instruction that correspond to frequency-advance, frequency-delay, and steady-state regions from Figure 4(b), respectively. Gaps in the genome show where instructions have been elided for clarity.

5 Conclusion

We have shown that digital evolution can evolve cooperative synchronization behaviors that are similar to behaviors observed in natural systems and the corresponding algorithms proposed recently for use in self-organizing computational systems. The evolved solutions utilized an adaptive frequency strategy similar to the Ermentrout model that

altered their control flow based on a combination of sensed information and the organism's age. We have also shown that the evolution of synchronization is robust to message loss. This result demonstrates that digital evolution shows promise as a means to produce relatively complex cooperative behaviors from very simple fitness functions.

Acknowledgments. This work was supported in part by NSF Grants CCF-0750787, CCF-0820220, and CNS-0751155; U.S. Army Grant W911NF-08-1-0495; and by a Quality Fund Grant from Michigan State University.

References

1. Southwick, E.E., Moritz, R.F.A.: Social synchronization of circadian rhythms of metabolism in honeybees (apis mellifera). Physiological Entomology **12**(2) (1987) 209–212
2. Backwell, P., Jennions, M., Passmore, N., Christy, J.: Synchronized courtship in fiddler crabs. Nature **391**(6662) (1998) 31–32
3. Holldobler, B., Wilson, E.O.: The Ants. Harvard University Press (1990)
4. Buck, J.: Synchronous Rhythmic Flashing of Fireflies, II. The Quarterly Review of Biology **63**(3) (1988) 265–289
5. Mirollo, R.E., Strogatz, S.H.: Synchronization of pulse-coupled biological oscillators. SIAM Journal on Applied Mathematics **50**(6) (1990) 1645–1662
6. Ermentrout, B.: An adaptive model for synchrony in the firefly *pteroptyx malaccae*. Journal of Mathematical Biology **29**(6) (1991) 571–585
7. Babaoglu, O., Binci, T., Jelasity, M., Montresor, A.: Firefly-inspired heartbeat synchronization in overlay networks. In: Proceedings of Self-Adaptive and Self-Organizing Systems (SASO). (2007) 77–86
8. Ofria, C., Wilke, C.O.: Avida: A software platform for research in computational evolutionary biology. Journal of Artificial Life **10** (2004) 191–229
9. Nowak, M.A.: Five rules for the evolution of cooperation. Science **314**(5805) (2006) 1560–1563
10. Floreano, D., Mitri, S., Magnenat, S., Keller, L.: Evolutionary conditions for the emergence of communication in robots. Current Biology **17**(6) (2007) 514–519
11. Marocco, D., Nolfi, S.: Self-organization of communication in evolving robots. In: Proceedings of the Conference on Artificial Life (ALIFE). (2006) 178–184
12. Teuscher, C., Capcarrere, M.S.: On fireflies, cellular systems, and evolware. Evolvable Systems: From Biology to Hardware **2606** (2003)
13. Ofria, C., Adami, C.: Evolution of genetic organization in digital organisms. In: Proceedings of DIMACS Workshop on Evolution as Computation. (1999)
14. Knoester, D.B., McKinley, P.K., Beckmann, B.E., Ofria, C.: Directed evolution of communication and cooperation in digital organisms. In: Proceedings of the European Conference on Artificial Life (ECAL). (2007)
15. Beckmann, B.E., McKinley, P.K., Ofria, C.: Evolution of an adaptive sleep response in digital organisms. In: Proceedings of the European Conference on Artificial Life (ECAL). (2007) 233–242
16. Knoester, D.B., McKinley, P.K., Ofria, C.: Cooperative network construction using digital germlines. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). (2008)
17. Knoester, D.B., McKinley, P.K.: Evolving virtual fireflies. Technical Report MSU-CSE-08-9, Computer Science and Engineering, Michigan State University, East Lansing, Michigan (May 2008) `http://www.cse.msu.edu/thinktank/firefly.pdf`.