Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Boston, Mass., USA, July 9-11, 2007.

Evolution of Cooperative Information Gathering in Self-Replicating Digital Organisms [*]

Benjamin E. Beckmann, Philip K. McKinley, David B. Knoester, and Charles Ofria

Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
E-mail: {beckma24,mckinley,dk,ofria}@cse.msu.edu

Abstract

We describe a study in the application of digital evolution *to the problem of cooperative information gathering. In digital evolution, self-replicating computer programs evolve to perform tasks and optimize resource usage in order to survive within a user defined computational environment. Instruction-level mutations during replication and CPU-cycle rewards for desired behavior produce natural selection within the population. The evolution is open-ended and not limited by human preconceptions. The contributions of this work are (1) to demonstrate that cooperative information gathering can evolve in digital organisms and (2) to provide insight into the fundamental processes governing evolution of such behavior.*

Keywords: *digital evolution, cooperative behavior, natural selection, data collection, sensor network.*

1. Introduction

As computing systems increasingly interact with the physical world, for example through wireless sensor networks and embedded systems, they are often required to operate in extreme situations. To remain effective, these systems must adapt to dynamic network conditions, conserve energy, compensate for hardware and software failures, fend off attacks, and optimize performance, all with minimal human intervention [1, 2]. Moreover, achieving these goals typically requires interaction and cooperation among multiple nodes, some of which may be malfunctioning or physically compromised. The complexity of designing robust computational systems has led many researchers to investigate *biologically-inspired* approaches. Natural organisms have an amazing ability to adapt to unforeseen circumstances. This adaptability can be codified in computational systems either through biomimetics [3], which mimics behaviors found in natural organisms, or through evolutionary computation [4–6], which models the natural processes that produce those behaviors. Both approaches have shown considerable promise in supporting the design of robust distributed systems [7–19].

Our work investigates the role *digital evolution* [20] might play in designing robust distributed systems. Digital evolution is a form of evolutionary computation in which populations of self-replicating computer programs, referred to as *digital organisms*, evolve to perform tasks within a user defined computational environment. The "genome" of a program is its sequence of instructions, typically drawn from an instruction set exhibiting certain robustness properties (see Section 2). When they replicate, digital organisms are exposed to instruction-level mutations. The resulting variation in genomes, combined with differential fitness to their environment (in the system we use here, organisms are rewarded with additional CPU cycles for exhibiting desired behaviors), produces natural selection within the population. Tracing the evolution of the digital organisms can provide insight into the evolutionary process, often revealing unexpected and strikingly clever solutions [21].

We are conducting a set of studies that explore how evolution can be "harnessed" to aid in the design of robust communication services. In an earlier study [22], we demonstrated that populations of digital organisms can evolve the ability to perform leader election. In this paper we report on a set of experiments that address the evolution of cooperative information gathering. The main contributions of this work are (1) to demonstrate that such behavior can evolve in digital organisms and (2) to provide insight into the fundamental processes governing evolution of this be-

* This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, National Science Foundation grants EIA-0130724, ITR-0313142, and CCF 0523449, and a Quality Fund Concept grant from Michigan State University.

havior. Combined, these studies help to identify which local behaviors enable complex global behaviors to *emerge* in the population. Effectively, this approach provides researchers with a "digital Petri dish" in which solutions to complex problems can evolve. In the case of communication operations, algorithms produced through digital evolution can be codified and evaluated using both traditional simulation techniques as well as experimentation on sensor hardware. These activities constitute our ongoing work, discussed later.

The remainder of this paper is organized as follows. Section 2 provides background on AVIDA [23], the digital evolution platform used in this study. Section 3 describes how, during the evolutionary process, rewards are given to individuals for performing a set of basic tasks, which lead to self-organizing information gathering behavior. Experimental results are presented in Section 4, followed by concluding remarks in Section 5.

2. Background

Evolutionary computation (EC) is a broad field focused on applying the basic principles of Darwinian evolution to solving optimization problems. The most well-known EC method is the genetic algorithm (GA) [4], an iterative search technique in which the individuals in the population are encodings of candidate solutions to an optimization problem. In each generation, the fitness of every individual is calculated, and a subset of individuals are selected, recombined and/or mutated, and moved to the next generation. Genetic programming (GP) [5, 6] is a related method where the individuals are actual computer programs. Both approaches have been successfully used to solve complex problems, even producing patentable designs [24].

Digital Evolution. While evolutionary computation has been studied since the 1960's, the specific field of digital evolution is much younger. The first experiments with populations of self-replicating computer programs were performed in 1990 in a system called Coreworld [25], and later improved upon in Tierra [26]. In 1993, Ofria and colleagues began development of AVIDA [20, 23], currently the most widely used digital evolution platform and the one used in this study. AVIDA was developed primarily to provide a better understanding of evolution in nature. Observing evolution in digital organisms enables scientists to address questions that are impossible to study with organic life forms, for example, by explicitly disabling whole categories of mutations and seeing how that constraint affects the population. However, unlike mere numerical simulations and other evolutionary computation techniques, digital organisms possess the ability to evolve in an open-ended fashion: whether a given organism self-replicates and moves into the next generation depends on its environment and its interaction with other organisms.

Over the past several years, AVIDA has been used to conduct pioneering research in the evolution of biocomplexity, with an emphasis on understanding the evolutionary design process [21, 27–29]. However, the search power of AVIDA can also be used to address complex problems in science and engineering [22, 30–32]. In this paper we demonstrate that a self-organizing, information gathering system can evolve in an environment where individuals die and are replaced continuously. In a later phase of this study we will port evolved algorithms for this and other operations to an experimental sensor testbed for evaluation. While evolved solutions may share the inherent imperfections of natural organisms, they might also be resilient to unexpected conditions, where human-designed algorithms are limited and/or brittle.

AVIDA Operation. Figure 1 depicts the structure of an AVIDA population and an individual digital organism (Avidian). Each Avidian comprises a circular list of assembly-like instructions (its genome) and a virtual CPU, and each occupies a *cell* within a virtual environment. AVIDA instructions enable an organism to perform simple mathematical operations, such as addition, increment, and bit-shifts, as well as interact with the organism's environment, for example, by sending a message to a neighboring organism, or outputting a number to the environment. The execution of instructions by an Avidian's CPU controls the behavior of the organism, including replication of its offspring. The architecture used in the study described here contains three general-puropse 32-bit registers (AX, BX, CX), two general-purpose stacks, and four special-purpose *heads* (READ-HEAD, WRITE-HEAD, FLOW-CONTROL-HEAD, and INSTRUCTION-HEAD). The READ-HEAD and WRITE-HEAD are used solely for self-replication, while the other two are used to allow non-sequential execution flow and as an instruction pointer.

Self-replication is a critical aspect of digital evolution, and distinguishes it from other forms of evolutionary computation. To avoid extinction, an organism must be able to produce offspring. This means that the instructions comprising the organism's genome must contain a sequence that will create a copy of the genome and split it off. In AVIDA, doing so requires the organism to copy its own instructions one-by-one to a newly allocated space and then issue an H-DIVIDE instruction. Each copy operation is subject to random mutations that can change the instruction, insert an extra instruction, or delete the instruction. A key property of AVIDA's instruction set that differs from traditional computer languages, is that it is not possible to construct a *syntactically incorrect* genome [30]. Hence, while random mutations will produce many genomes that do not

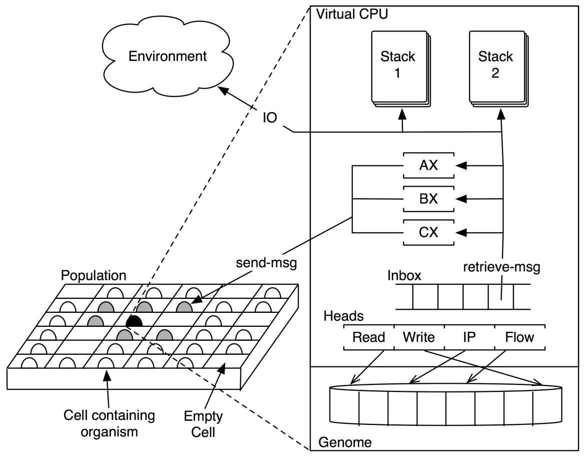


Figure 1. An AVIDA population (left) and the structure of an Avidian (right)

perform any meaningful computation, their instruction sequences will still be valid. After replication, the offspring is placed in another cell, terminating any organism already residing there. Each AVIDA run begins with a single default organism capable of only replication; this organism is an ancestor to every organism present in the run.

AVIDA Environment. In AVIDA, digital organisms compete against each other for survival. Specifically, they compete for space within a fixed population that is confined to a planar grid of cells, shown bottom left of Figure 1. Avidians can gain a competitive advantage through mutations and outcompete other Avidians by producing offspring faster. Therefore, it is likely that more copies of a highly fit Avidian's genes are present in the population relative to less fit Avidians, pushing the latter toward extinction. To gain a competitive advantage, Avidians either (1) become more efficient replicators through code optimization, or (2) perform user defined tasks. A *task* in AVIDA is a mechanism to reward or punish the offspring of an Avidian that has successfully performed a specific function during its lifetime. Tasks are defined in terms of an organism's observable behavior, or *phenotype*, which manifests itself in two ways: IO between an Avidian and the environment, and messages sent between two Avidians. Upon replication, the offspring of an organism that has completed a task receives a bonus in terms of CPU cycles, increasing its own chance of replicating successfully.

Avidians exchange information by sending messages to neighbors. Table 1 describes messaging instructions. The SEND-MSG instruction sends a message to a neighboring cell. Each Avidian has a *facing*, a pointer to one of its neighbors, used to determine the recipient of a message. An Avidian can change its facing by executing the ROTATE-TEMPLATE instruction, which rotates an organism to the direction specified by a *template* (a sequence of no-operation instructions [30] following the instruction). When a message arrives, it is placed into the recipient's inbox, shown in Figure 1, for later retrieval. An Avidian can read a message from its inbox into its virtual CPU with the RETRIEVE-MSG-ZERO instruction. If no message is available, zeros are written to the designated registers.

Table 1. Communication-related instructions

Instruction Name	Default behavior
SEND-MSG	$Send(BX, CX)$.
RETRIEVE-MSG-ZERO	If $\exists\, msg$ **then** $BX \leftarrow msg_{label}$, $CX \leftarrow msg_{data}$ **else** $BX \leftarrow 0$, $CX \leftarrow 0$
GET-ID	$BX \leftarrow$ Cell ID.
ROTATE-TEMPLATE	Rotate the facing of the organism to direction specified by template.

3. Problem Description

The abundance of self-organizing behaviors in natural systems leads to questions as to how these behaviors evolved. Ants, bees, and birds all exhibit behaviors that are both self-organizing and cooperative [3]. Many classical distributed algorithms such as leader election, data gathering, consensus, and load balancing, also require system level behaviors similar to those found in nature. For this reason, researchers have investigated biologically-inspired approaches to designing computational systems. Examples include mimicking the social behavior of insect colonies in robotic foraging [3, 33, 34], using chemotaxis to facilitate robust network routing [9], modeling colony behaviors in consructing highly-available services [7] and applying the concept of emergent behavior to the design of robust protocols and distributed systems design [8, 10, 35–38].

However, while such biomimetic approaches have achieved considerable success, they are limited to codifying behaviors observed in nature today. In fact, those behaviors were produced through millions of years of evolution. In other words, biomimetic approaches seek to imitate the *results* of evolution, but do not exploit the power of natural selection to produce solutions that best fit the environment. For example, we might design the control software on a micro-robot so that it mimics behaviors found in ants. However, while the robot may possess some physical characteristics reminiscent of an ant, *it is not an ant*. On the other hand, if we had the ability to *evolve* the control software, taking into account the capabilities of the robot and the characteristics of its environment, we might discover behaviors that are more effective in controlling the robot and completing tasks, than imitated ant behavior. Indeed, the evolved behaviors might not exist in *any* living creature. Digital evolution gives us this power.

Our research uses AVIDA to study the evolution of cooperative behavior. We have previously demonstrated the ability of Avidians to perform leader election [22] and adapt to the environment in order to conserve energy [39]. Solutions evolved *off-line* can be deployed in simulators and on actual hardware (the programs do to not continue to evolve once deployed). We are currently developing a tool to translate AVIDA genomes to code that can execute directly on motes. In this paper we focus on a particular type of cooperation needed in sensor networks and other distributed systems, namely, gathering information from multiple locations and delivering the data to a collection point ("sink" node). Figure 2 depicts the desired behavior in the context of AVIDA. Here, we define sensed data as a unique 32-bit random ID that is statically assigned to each cell. To gain access to a cell's ID, the occupying Avidian must execute the GET-ID instruction. An Avidian reading its cell ID is analogous to the sensing of an external stimulus local to a sensor. Next, the cell ID must be delivered to the collection point by the population of Avidians. A successful journey depends on the level of cooperation within the population. The solution must be highly robust, overcoming continuous turnover in the population that results in lost messages and terminated organisms.

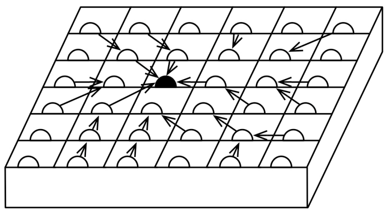


Figure 2. Desired information gathering behavior with black semicircle denoting the collection point.

AVIDA enables the user to define a set of (often simple) tasks and observe how evolution combines them to produce more complex behaviors. In this study, we extended AVIDA's task library to include several communication-oriented tasks listed in Table 2. The tasks in Table 2 are divided into two sections; tasks that reward organisms for sending cell IDs (upper), and tasks that reward organisms for efficient directional message sending (lower). The "linear" tasks reward the sending of IDs not sent previously by the organism; these will be discussed later. The tasks in the lower set place a selective pressure on the population to send messages toward the collection point, by rewarding organisms for sending messages that make progress toward the collection point, and punishing those that send messages away from the collection point. Progress is determined using minimum hop count between nodes. Hop count was chosen because the total amount of energy used to send a message to the collection point is proportional to the number of hops the message traverses on its way to the collection point. Section 4 describes the global information gathering behavior evolved as a result of integrating these tasks into AVIDA.

Table 2. Tasks to promote information gathering behavior.

Task Name	Rewarded Sending Behavior
SENT-SELF	own cell ID
SENT-ID-NOT-SELF	cell IDs other than own
SENT-ID-LINEAR	unique cell ID
SENT-ID-NOT-SELF -LINEAR	unique cell ID other than own
SENT-TOWARD	message toward collection point
SENT-NOT-TOWARD	message not toward collection point (negative reward)
SENT-ID-TOWARD	cell ID toward collection point

To better illustrate how AVIDA uses tasks, let us consider the SENT-SELF task, which rewards an organism for sending its cell ID as the data field of a message. Table 3 shows an example of AVIDA code that completes the SENT-SELF task. The organism executing this code obtains its cell ID, which it sends to a neighbor as the data field of a message. The code in Table 3 is handwritten, however as will be shown in Section 4.2, the evolved code produced by AVIDA is comparable. Algorithm 1 gives pseudo-code for the AVIDA function that checks for completion of the SENT-SELF task. This code is executed whenever an organism sends a message. If the function returns 1.0, the calling organism will be credited. If the organism replicates in the future, its offspring will receive a bonus in the form of virtual CPU cycles.

Throughout an experiment only limited global information about the AVIDA environment is available to the population directly. For example, the location of a collection point within the population must be discovered through the evolutionary process. The only environmental knowledge an organism has access to is its position in a Cartesian co-

Table 3. Sample AVIDA code that reads an organisms cell ID into its virtual CPU's CX register and then sends the contents of the CX register as the data field of a message to the northwest.

#	Instruction	Description
1	GET-ID	$CX \leftarrow$ Cell ID
2	NOP-C	
3	ROTATE-TEMPLATE	face northwest
4	NOP-B	
5	SEND-MSG	$Send(BX, CX)$

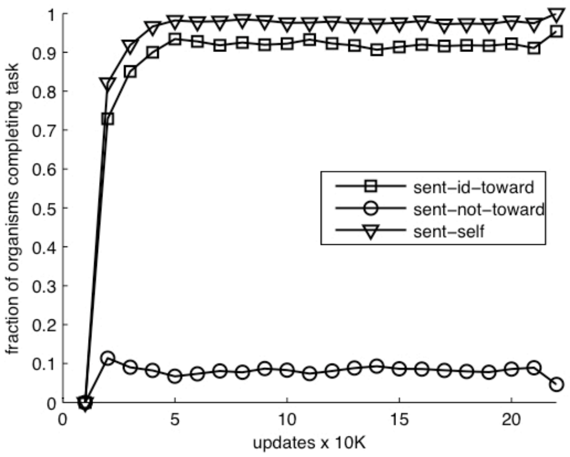
Algorithm 1 Task-SentSelf(TaskContext ctx): This task rewards an organism for sending a message where the data field of the message contains the sender's ID.

```
msg ← ctx.GetMessage()
if msg = NULL then
  return 0.0
end if
if msg.GetData() = msg.GetSender().GetCellID() then
  return 1.0
end if
return 0.0
```

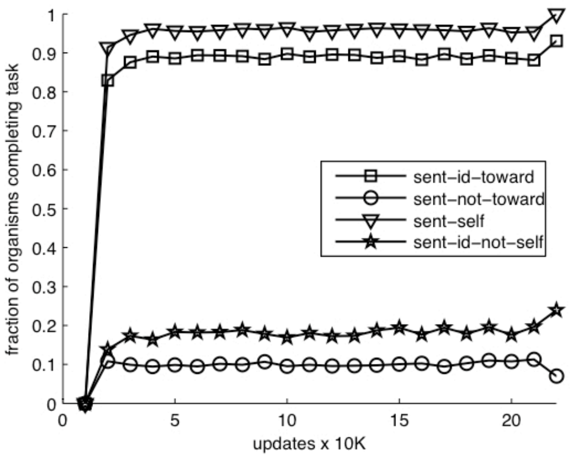
ordinate space, which is obtained through the use of the GET-POS instruction. It should be noted that the evolutionary process is also solely responsible for finding and coordinating instruction usage. We do not provide any prior knowledge regarding an instruction's usage or operand(s). We seed the population with an organism capable of only self-replication. The usage of an instruction and its combination with other instructions to achieve a task must be evolved.

4. Experimental Results

Evolving a information gathering application in AVIDA using the instructions and tasks described previously is very much an experimental process. The task configurations described here represent only a small sampling of the experiments conducted. We describe three sets of experiments: 1) initial evolution of information gathering behavior; 2) elimination of undesirable individual behavior; and 3) using a mobile collection point. In each experiment a population of 625 organisms was organized in a 25×25 grid and allowed to evolve for 200,000 updates. An *update* is a measure of time within AVIDA. In this study, an average organism executes 30 instructions per update. After update 200,000, mutations were turned off, and the populations were allowed to stabilize over another 20,000 updates.



(a) initial results



(b) addition of SENT-ID-NOT-SELF task

Figure 3. Task completion in initial tests.

4.1. Initial Runs

The goal of our initial experiment is to evolve Avidians that send their cell IDs through the network of organisms to the collection point. Configuring AVIDA and initiating runs using the tasks with static rewards, described in Table 2, we observed the beginnings of information gathering behavior. Figure 3 shows the results of two separate experiments formulated as a proof-of-concept. In the runs that correspond to Figure 3(a), we rewarded organisms for sending their own cell ID in the direction of the collection point, through the use of the SENT-ID-TOWARD and SENT-SELF tasks. We also discouraged sending messages that did not make progress toward the collection point by punishing individuals who completed the SENT-NOT-TOWARD task. In the runs that correspond to Figure 3(b), we added the SENT-ID-NOT-SELF task to reward organisms for forwarding received IDs. This task is highly dependent on neighboring

Avidians sending valid cell IDs because Avidians have no way of determining whether an ID is valid. The results in both Figure 3(a) and 3(b) plot the fraction of organisms that have achieved each task, averaged over ten runs. Figure 3(a) clearly shows that the SENT-ID-TOWARD and SENT-SELF tasks are performed by a large majority of the population. Further, once mutations were turned off at update 200,000, 100% of the organisms in the population were sending their own cell ID, while approximately 95% were sending messages toward the collection point.

Although Figure 3(b) is similar to Figure 3(a), the underlying behaviors in these sets of runs differ in an important way. Both sets of runs produced messages containing cell IDs that were destined for the collection point. For Figure 3(a) the collection point received 37.2 unique cell IDs, on average, with a standard deviation of approximately 3.74. However, with the addition of the SENT-ID-NOT-SELF task, in Figure 3(b), this number increased to 132.8, on average, with a standard deviation of approximately 12.59. The nearly four-fold increase in ID collection was caused solely by the SENT-ID-NOT-SELF task, which promoted ID forwarding.

Along with the cooperative behavior needed to deliver cell IDs to the collection point, we also observed selfish behavior within the population. One example of this behavior arose in multiple runs represented in Figure 3(a). Not rewarding the forwarding of received IDs allowed Avidians to repeatedly send their own cell ID without checking for a received message, effectively eliminating ID forwarding. Even with the addition of the SENT-ID-NOT-SELF task, selfish behavior evolved in some runs. Specifically, Avidians evolved to store, on their stack, a subset of the messages they received, repeatedly forwarding only the stored subset. This strategy was successful because they were likely to store and forward another Avidian's cell ID, while many other messages, possibly containing valid IDs, remained in their inbox and were never forwarded.

4.2. Linearly Increasing Rewards

To reduce the selfish behavior described above, we introduced ID-sending tasks whose rewards increase linearly with the number of uniquely forwarded cell IDs. Replacing the SENT-ID-NOT-SELF task with the SENT-ID-NOT-SELF-LINEAR task offers an incentive for individuals to continually check their inbox. Figure 4 shows the results of this change with respect to task completion. Comparing Figure 3(b) and Figure 4 reveals no significant difference, so an expectation of similar performance in collecting cell IDs is expected. Indeed, the average number of IDs collected in the Figure 4 runs is 113.2 with standard deviation of approximately 6, which is a small decline in the collection totals. The number of messages sent within the population, however, is significantly reduced, as shown in Figure 5. Specifically, the replacement of the SENT-ID-NOT-SELF task with the SENT-ID-NOT-SELF-LINEAR task reduced the total number of messages by approximately 25%. In a sensor network, this decrease in network traffic would represent a large energy savings. We also combined the SENT-SELF and SENT-ID-NOT-SELF-LINEAR into the SENT-ID-LINEAR task and obtained similar results.

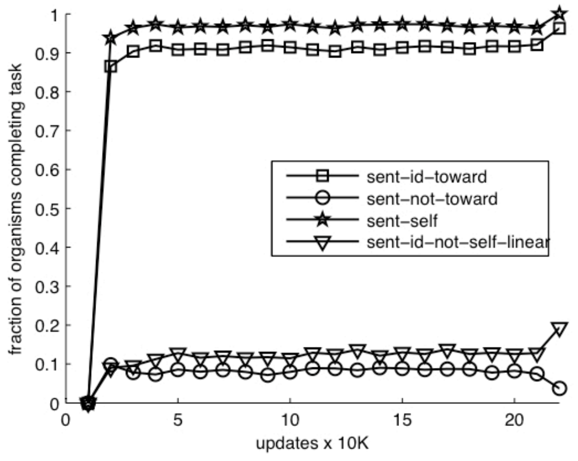


Figure 4. Runs replacing SENT-ID-NOT-SELF task with SENT-ID-NOT-SELF-LINEAR task.

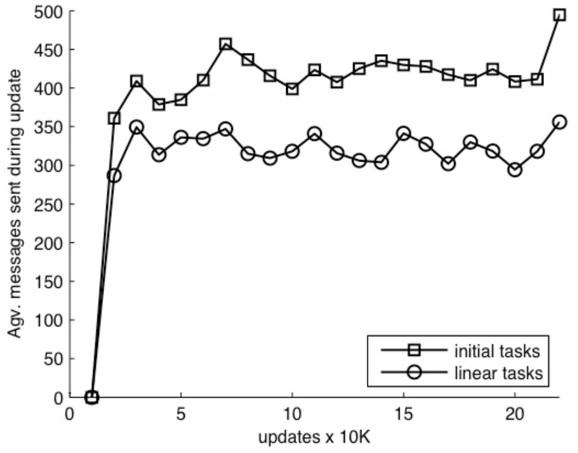


Figure 5. Average number of messages sent in runs from Figures 3(b) and 4

Table 4 shows a sample of a genome evolved in one of the runs from Figure 4. Three sections of the genome are shown, each corresponding to a critical behavior produced through evolution. Lines $2-7$ show the evolved instruction sequence used to read the organism's cell ID, rotate to face the collection point, and send its cell ID. As shown in Table 3, a handwritten solution that performs equivalent behavior can be written in only one less line. Lines $9 - 13$ show the evolved instruction sequence used to forward a received message. If a message is present in the Avidian's inbox it is retrieved and then forwarded, possibly switching the order of the message's contents. Finally, lines $15 - 23$ show the organism's evolved copy loop. By including multiple H-COPY instructions in the copy loop, the organism replicates with fewer executed instructions. Most of the code in Table 4 could be mapped onto the virtual machine of a sensor node with little modification. We plan to extend this work by encoding an evolved genome on a set of wireless sensor devices and empirically testing the system's effectiveness under real-world conditions.

Table 4. Portion of evolved genome.

#	Instruction	Description
1	...	
2 3	GET-ID NOP-C	$CX \leftarrow$ Cell ID
4	SUB	$BX \leftarrow BX - CX$
5 6	ROTATE-TEMPLATE NOP-B	face northwest
7	SEND-MSG	$Send(BX, CX)$
8	...	
9	RETRIEVE-MSG-ZERO	$BX \leftarrow inbox.front().label$ $CX \leftarrow inbox.front().data$
10 11	IF-LABEL NOP-A	If NOP-A was last copied execute next inst., else skip
12	SWAP	$BX \leftrightarrow CX$
13	SEND-MSG	$Send(BX, CX)$
14	...	
15	H-SEARCH	FLOW-HEAD $\leftarrow$ inst. 16
16 17 18	H-COPY H-COPY H-COPY	copy instruction from READ-HEAD to WRITE-HEAD
19 20	IF-LABEL NOP-C	If NOP-C was last copied execute next inst., else skip
21	H-DIVIDE	split organism into two
22	H-COPY	
23	MOV-HEAD	INSTR.-HEAD $\leftarrow$ inst. 16
24	...	

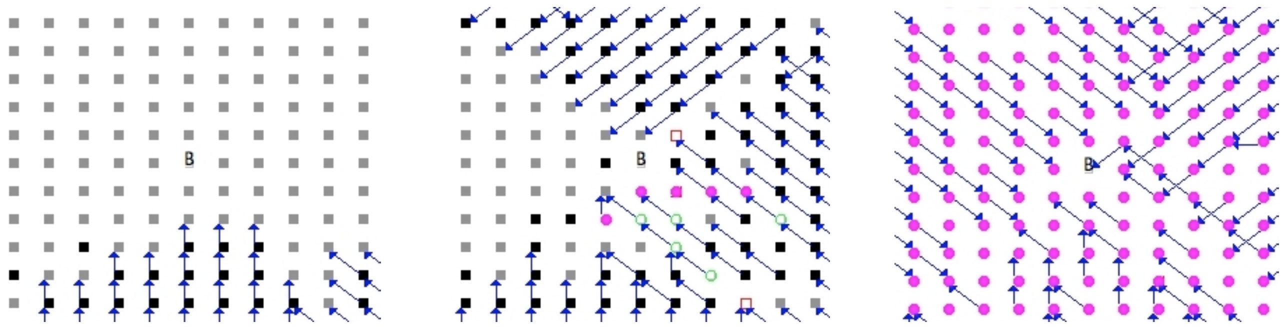
Figure 6 depicts the behavior of the population in one of the runs from Figure 4. Each subfigure shows the state of a small area (11×11) of the population (25×25), centered on the collection point (labeled 'B') at varying times during evolution. (The evolution of the full population can be viewed at http://www.cse.msu.edu/∼mckinley/avida.) A solid gray square represents the existence of an Avidian that has not sent a message prior to the end of the update, whereas, a solid black square represents an Avidian that has done so. A line emanating from a solid black square represents a sent message that made progress toward the collection point. A solid black square with no line emanating from it represents an organism that has sent a message that did not make progress toward the collection point. A hollow square represents an Avidian that has sent a message to a cell which was not occupied when the message was sent.

Figure 6(a) shows a subsection of the population a short time after the run started. The grid has filled up with organisms, denoted by the presence of squares, and a subset of Avidians, located on the bottom of the grid, has begun to send messages toward the collection point. Figure 6(b) shows this trend continuing and, for the first time, cells whose IDs have been communicated to the collection point: A hollow circle denotes a cell whose ID has been received at the collection point for the first time during this update, and a solid circle represents a cell whose ID was previously received by the collection point. Figure 6(c) shows the population of Avidians after 2,000,000 updates. At this time point 337 out of 625, or $\approx 54\%$, of the cell IDs have been collected. The rather low percentage of cell IDs collected is due to the high turnover rate, roughly 5% per update, within the population, which causes the loss of messages containing valid IDs when an Avidian holding these messages is overwritten. Even with a high turnover rate, however, more than half of the cell IDs were collected. It should be noted that with continued evolution the number of random IDs reaching the collection point continues to increase.

4.3. Moving the Collection Point

In the next set of experiments, we modified the runs described above by relocating the collection point periodically to determine if Avidians could adapt to the change. As shown in Figure 7, the collection point initially starts in the center of the population, position $(12, 12)$. After 50,000 updates the collection point is relocated to position $(0, 0)$, where it remains until update 100,000. At update 100,000 the collection point is again relocated, this time to position $(24, 24)$, where it remains for the next 50,000 updates. The collection point is moved to it final position $(12, 12)$ at update 150,000.

Analyzing the data from these runs reveals evidence of a self-modifying network within the AVIDA population. Figure 8 plots the task completion fraction, which is similar to that in Figures 3(b) and 4. The spikes present in Figure 8 correspond to the fraction of organisms completing the SENT-ID-TOWARD and SENT-NOT-TOWARD tasks just after the collection point is moved. The center spike is much larger than the outer spikes because that data point occurs just after the collection point is moved from the northwest corner of the population to the southeast corner. When this



(a) update 500 (b) update 2,000 (c) update 2,000,000

Figure 6. Subsection (11×11) of the population (25×25) showing evolution of information gathering.

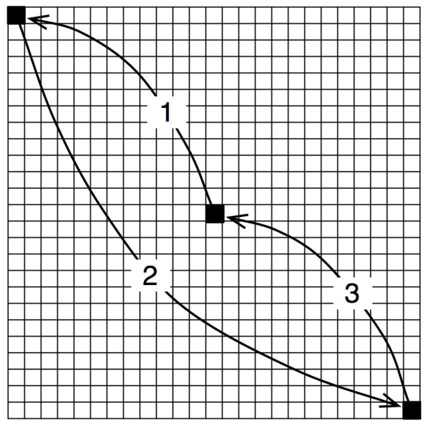


Figure 7. Movement of collection point

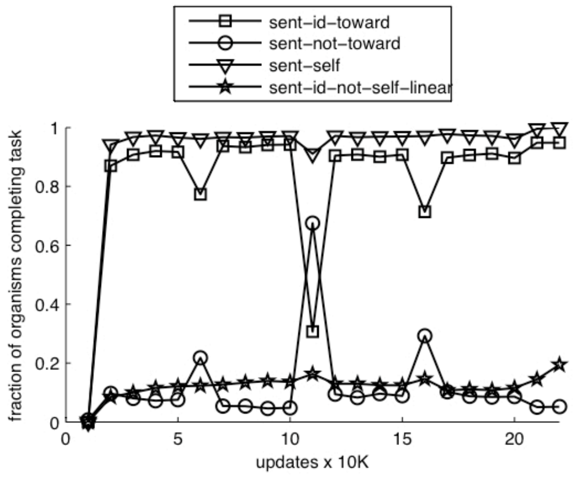


Figure 8. Results of moving collection point.

```
1  ...
2  GET-ID
3  ROTATE-TEMPLATE
4  NOP-B
5  SEND-MSG
6  ...
```

```
1  ...
2  GET-ID
3  ROTATE-TEMPLATE
4  NOP-C
5  SEND-MSG
6  ...
```

Figure 9. Mutation of template causing message to be send in a different direction.

move occurs the entire population must adapt to send messages southeast, where as just prior to the move the majority of the messages in the population were being sent northwest. We also observed that the quantity of messages sent within the population does not increase substantially compared to the runs from Section 4.2, indicating the organisms adapted fairly quickly to the new environment.

It should be noted that natural selection drives the population's ability to adapt and find the new location of the collection point. Prior to the initial relocation of the collection point, organisms have already built up the functionality within their genome to send and forward ID carrying messages. The left side of Figure 9 shows a sequence of instructions that will send an organism's cell ID to the northwest. A mutation of line 4, shown in right side of Figure 9, causes the organisms to rotate to face southeast. Therefore, the rediscovery of the collection point is quicker than its initial discovery, because only the template following an instance of the ROTATE-LABEL instruction needs to be mutated for the data collection behavior to reappear.

5. Conclusions and Future Directions

We have used the AVIDA digital evolution platform to show that it is possible for a population of digital organisms to evolve self-organizing, information gathering behavior. Observing the evolutionary process provides insight into how to design new protocols and algorithms for highly dynamic environments. Evolved protocols may be resilient to adverse conditions, due to the population turnover rate and the environment's unpredictability.

Using digital evolution to design software is not without its limitations. Even though the language used in AVIDA is Turing complete, it is very low-level, and the virtual CPU used here is very simple: a single organism can store only 23 numbers, either in registers or on stacks. More complex architectures and instruction sets are under development. However, even with these limitations, AVIDA produced solutions to the problem of self-organizing information gathering.

We intend to extend this work into simulation and real-world sensor devices. We are beginning work on extending the AVRORA [40] simulator to handle AVIDA-evolved output. Using AVRORA, we plan to assess the effectiveness of AVIDA-evolved solutions to reliably gather information in real world situations. We are also studying the addition of mobility, using a system called THINKTANK. Specifically, we have recently developed an instruction set that includes simple motor control primitives and sensors. We expect to use this platform to evolve individuals that use these new features in order to traverse obstacle courses, elude predators, and catch moving targets.

The true benefit of evolving software in an artificial life system like AVIDA is the high-level of abstraction that it provides. Stating and implementing tasks that model behaviors of a system is an improvement over hand-coded implementation in a high-level programming language. Using this technique, a developer can work at the application level, focusing on high-level behaviors rather than lower-level concerns. If this technique proves to be successful in deployed networks, as technology allows, it will be applied to medium and large scale computing systems. Our early results are promising and indicate that evolving application level behaviors through artificial life is worthy of further investigation.

Further Information. Papers on digital evolution and the AVIDA software are available at `http://devolab.cse.msu.edu`. Information on evolving cooperative behavior can be found at `http://www.cse.msu.edu/~mckinley/avida`.

Acknowledgments. The authors gratefully acknowledge the contributions of the following individuals to this work: Jeffrey Clune, Sherri Goings, David Bryson, Richard Lenski, Heather Goldsby, and Betty Cheng.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[3] S. Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, 2002.

[4] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[5] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1st ed., 1989.

[6] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Genetic Programming, Springer, 1st ed., 2005.

[7] J. Suzuki and T. Suda, "A middleware platform for a biologically-inspired network architecture supporting autonomous and adaptive applications," in *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Intelligent Services and Applications in Next Generation Networks*, February 2005.

[8] R. J. Anthony, "Emergence: a paradigm for robust and scalable distributed applications," in *1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA*, IEEE, 2004.

[9] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, 2006.

[10] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," in *Proceedings of the 3rd International Workshop on Engineering Self-Organising Applications (ESOA 2005)*, (Utrecht), July 2005.

[11] S. Pleisch, M. Balakrishnan, K. Birman, and R. van Renesse, "Mistral: Efficient flooding in mobile ad-hoc networks," in *Proceedings of the Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM MobiHoc 2006)*, (Florence, Italy), May 2006.

[12] M. Mamei and F. Zambonelli, "Programming stigmergic coordination with the TOTA middleware.," in *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pp. 415–422, 2005.

[13] G. Di Caro, F. Ducatelle, and L. Gambardella, "Swarm intelligence for routing in mobile ad hoc networks," in *Proceedings of the IEEE Swarm Intelligence Symposium*, (Pasadena, California, USA), November 2005.

[14] M. Guimarães and L. Rodrigues, "A genetic algorithm for multicast mapping in publish-subscribe systems," in *NCA '03: Proceedings of the Second IEEE International Symposium on Network Computing and Applications*, (Washington, DC, USA), IEEE Computer Society, 2003.

[15] A. Abraham and C. Grosan, "Evolving intrusion detection systems," in *Genetic Systems Programming: Theory and Experiences* (N. Nedjah, A. Abraham, and L. de Macedo

Mourelle, eds.), vol. 13 of *Studies in Computational Intelligence*, pp. 57–80, Germany: Springer, 2006.

[16] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[17] S. Pierre and G. Legault, "A genetic algorithm for designing distributed computer network topologies," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 28, pp. 249–258, Apr 1998.

[18] A. Koyama, T. Nishie, J. Arai, and L. Barolli, "A new quality of service multicast routing protocol based on genetic algorithm," in *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, (Washington, DC, USA), pp. 655–660, IEEE Computer Society, 2005.

[19] P. Dasgupta, "Intelligent agent enabled genetic ant algorithm for P2P resource discovery.," in *Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing*, pp. 213–220, 2004.

[20] C. Ofria and C. O. Wilke, "Avida: Evolution experiments with self-replicating computer programs," in *Artificial Life Models in Software* (A. Adamatzky and M. Komosinski, eds.), pp. 3–35, Springer Verlag, London, 2005.

[21] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami, "The evolutionary origin of complex features," *Nature*, vol. 423, pp. 139–144, 2003.

[22] D. B. Knoester, P. K. McKinley, and C. A. Ofria, "Using group selection to evolve leadership in populations of self-replicating digital organisms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, July 2007. to appear.

[23] C. Ofria and C. O. Wilke, "Avida: A software platform for research in computational evolutionary biology," *Journal of Artificial Life*, vol. 10, pp. 191–229, 2004.

[24] J. R. Koza, M. A. Keane, M. J. Streeter, M. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Springer, 2003.

[25] S. Rasmussen, C. Knudson, P. Feldberg, and M. Hindsholm, "The coreworld: Emergence and evolution of cooperative structures in a computational chemistry," *Physica D*, vol. 42, no. 1-3, pp. 111–134, 1990.

[26] T. S. Ray, "An approach to the synthesis of life," in *Artificial Life II* (C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds.), pp. 371–408, Reading, MA, USA: Addison-Wesley, 1992.

[27] R. E. Lenski, C. Ofria, T. C. Collier, and C. Adami, "Genome complexity, robustness and genetic interactions in digital organisms," *Nature*, vol. 400, pp. 661–664, 1999.

[28] C. Adami, C. Ofria, and T. C. Collier, "Evolution of biological complexity," *Proc. Natl. Acad. Sci. USA*, vol. 97, pp. 4463–4468, 2000.

[29] C. O. Wilke, J. Wang, C. Ofria, C. Adami, and R. E. Lenski, "Evolution of digital organisms at high mutation rate leads to survival of the flattest," *Nature*, vol. 412, pp. 331–333, 2001.

[30] C. Ofria, C. Adami, and T. C. Collier, "Design of evolvable computer languages," *IEEE Transactions in Evolutionary Computation*, vol. 17, pp. 528–532, 2002.

[31] S. Chow, C. O. Wilke, C. Ofria, R. E. Lenski, and C. Adami, "Adaptive radiation from resource competition in digital organisms," *Science*, vol. 305, pp. 84–86, 2004.

[32] H. J. Goldsby, D. B. Knoester, B. H. Cheng, P. K. McKinley, and C. A. Ofria, "Digitally evolving models for dynamically adaptive systems," in *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, May 2007. to appear.

[33] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella, "Evolving self-organizing behaviors for a swarm-bot," *Autonomous Robots*, vol. 17, no. 2–3, pp. 223–245, 2004.

[34] R. Schoonderwoerd, J. L. Bruten, O. E. Holland, and L. J. M. Rothkrantz, "Pheromone robotics," *Autonomous Robots*, vol. 11, no. 3, pp. 319–324, 2004.

[35] J. Debenham, "A multiagent system manages collaboration in emergent processes," in *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, (New York, NY, USA), pp. 175–182, ACM Press, 2005.

[36] J. Cao, "Self-organizing agents for grid load balancing," in *5th International Workshop on Grid Computing (GRID 2004)*, (Pittsburgh, Pennsylvania, USA), pp. 388–395, IEEE/ACM, November 2004.

[37] M. Kubo, C. Dan, H. Sato, and T. Matubara, "An adaptive behavior of mobile ad hoc network agents," in *Proceedings of the 9th International Conference on Intelligent Autonomous Systems*, pp. 243–252, 2006.

[38] J. Rao and S. Biswas, "Controlled mobility: A mechanism for energy load distribution in sensor networks," in *Proceedings of GLOBECOM 2006*, November 2006.

[39] B. Beckmann, P. K. McKinley, and C. Ofria, "Evolution of an adaptive sleep response in digital organisms," Tech. Rep. MSU-CSE-07-19, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, April 2007.

[40] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," in *IPSN '05: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, (Piscataway, NJ, USA), IEEE Press, 2005.