Using Group Selection to Evolve Leadership in Populations of Self-Replicating Digital Organisms

David B. Knoester, Philip K. McKinley, and Charles A. Ofria
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{dk,mckinley,ofria}@cse.msu.edu

ABSTRACT

This paper describes a study in the evolution of distributed cooperative behavior, specifically leader election, through digital evolution and group 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. Group selection is the theory that the survival of the individual is linked to the survival of the group, thus encouraging cooperation. The results of experiments using the AVIDA digital evolution platform demonstrate that group selection can produce populations capable of electing a leader and, when that leader is terminated, electing a new leader. This result serves as an existence proof that group selection and digital evolution can produce complex cooperative behaviors, and therefore have promise in the design of robust distributed computing systems.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Self-modifying machines*

General Terms

Experimentation.

Keywords

Digital evolution, leader election, cooperative behavior, natural selection, group selection, mutation, autonomic computing, biologically-inspired computing.

1. INTRODUCTION

The increasing interaction between computing technology and the physical world requires that systems be able to adapt to changing conditions [24]. Adaptation and robust operation are especially important at the wireless edge of the Internet, where systems must tolerate lossy communication, conserve energy, compensate for failures, fend off attacks, and optimize performance, all with minimal human intervention. *Autonomic computing* [15] refers to systems capable of such self-management. To design robust computational systems, one can take inspiration from nature. Living organisms have an amazing ability to adapt to changing environments, both in the short term (phenotypic plasticity) and in the longer term (genetic evolution). Moreover, most complex organisms exhibit traits desirable in self-managing computing systems: *system monitoring* (senses, awareness); *system reconfiguration* (muscle growth, calluses); *self-repair* (blood clotting, tissue healing); and *intrusion detection/elimination* (immune systems).

Biologically-inspired approaches to system design include biomimetics [14], where computational systems mimic behaviors found in natural organisms, and evolutionary computation methods such as genetic algorithms, genetic programming, and digital evolution [10, 13, 17, 26], which codify the natural processes that produce those behaviors. Examples of biomimetics include mimicking the social behavior of insect colonies in robots [14,23,30] and using the concept of chemotaxis to facilitate robust network routing [2]. Genetic algorithms and genetic programming have also been applied to a variety of problems in distributed computing, including multicast mapping [11], multi-agent systems [6], and the automated design of communication protocols [38]. In addition, hybrid approaches have been proposed that use genetic programming to influence swarm dynamics [18], and neuroevolution has been used to study communication and cooperative behaviors [22, 33].

Our work addresses the application of *digital evolution* [26] to the design of robust distributed computing systems. In digital evolution, a population of computer programs exists in a user-defined computational environment. These "digital organisms" self-replicate, compete for available resources, and are subject to instruction-level mutations and natural selection. Over thousands of generations, they can evolve to survive, and even thrive, under extremely dynamic and adverse conditions. Unlike biomimetic approaches, digital evolution is not connected to behaviors found in existing living organisms. Moreover, digital evolution is open-ended: whether a given organism self-replicates and moves into the next generation depends on its environment and its interaction with other organisms. AVIDA [28], a digital evolution platform, has been used to study the evolution of biocomplexity in nature [1, 20] and to address complex problems in science and engineering [9, 19, 34].

In this paper, we use AVIDA to investigate whether group selection can produce a cooperative behavior, leader election, in populations of digital organisms. *Leader election* is a distributed algorithm whereby a population of processes must eventually elect a leader, and, if that leader is terminated, elect a new leader [21].

Group selection is the theory that the survival of the individual is linked to the survival of the group[1] [35] and is similar to multi-population evolutionary algorithms [4]. Multi-population approaches have recently been used to accurately diagnose malignancy in cancer [39] and improve runtime performance of multi-objective evolutionary algorithms [8]. Automatic grouping and role determination of individuals has also been applied to cooperative multi-agent problems [12, 25].

In AVIDA, group selection is realized by allowing distinct sub-populations (called demes) to evolve independently and to periodically compete against each other based on a fitness function. Results of our experiments demonstrate that this method can produce populations capable of electing a leader, and when that leader is terminated, electing a new leader. We emphasize that these digital organisms have no "built-in" ability to perform this task; each population begins with a single organism that possesses only the ability to self-replicate. Over thousands of generations, random mutations and natural selection produce an instruction sequence that realizes leader election. Furthermore, the group selection method used in our experiments selects only for the desired outcome, not for a particular implementation.

This study serves as an existence proof that group selection and digital evolution can produce complex cooperative behaviors. Our long-term goal is to use digital evolution in the design of robust distributed systems that remain effective even under extremely harsh conditions. While these 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. The remainder of this paper is organized as follows. Section 2 describes the AVIDA platform for digital evolution. Section 3 describes AVIDA's mechanism for studying group selection. Section 4 describes our experiments in evolving leader election, presents results from the use of AVIDA, and analyzes the genomes of a deme that evolved the desired behavior. Finally, Section 5 presents our conclusions and discusses our planned future work.

2. AVIDA BACKGROUND

In this section we provide an overview of AVIDA, including the structure of digital organisms and their environment.

2.1 Digital Organisms

Figure 1 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 "lives" in a common virtual environment. Within this environment, organisms may communicate with each other via the exchange of messages, resources may be produced and consumed, and organisms may sense and change properties of the environment. At any point in time the population of digital organisms may contain many different genomes. Some may be closely related (e.g., parent and offspring), while others may be related only through a distant ancestor.

Instruction set

An organism's genome is a circular list of instructions, similar in appearance and functionality to traditional assembly language instructions. The instructions enable an organism to perform simple mathematical operations, such as addition, multiplication, and

[1]The authors are aware of some controversy concerning group selection, kin selection, the evolution of altruism, and the "selfish gene" in *biological* systems. The research presented in this paper does not address this controversy. Instead, we use a clearly defined mechanism for group selection to evolve a specific behavior in digital organisms.

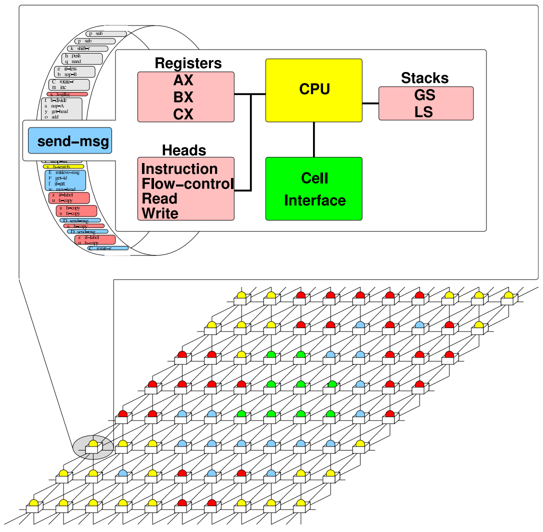


Figure 1: An AVIDA population containing 4 primary genomes (bottom), and the structure of an individual organism (top).

bit-shifts, as well as to interact with the organism's environment, for example, by sending a message to a neighboring organism, or outputting a number to the environment. The standard AVIDA instruction set is Turing-complete, and therefore theoretically able to evolve any computable function. A key property of AVIDA's instruction set that differs from traditional computer languages, however, is that it is not possible to construct a *syntactically incorrect* genome in AVIDA; that is, all possible genomes are "runnable." Hence, while random mutations will produce many genomes that do not perform any meaningful computation, their instruction sequences will still be valid. Indeed, this property is critical to the evolutionary process [27].

Instructions are executed by the organism's virtual CPU. Different AVIDA CPU architectures have been implemented and used in various studies [28]. The architecture used in this study contains a circular list of three general-purpose registers $\{AX, BX, CX\}$, two general-purpose stacks $\{GS, LS\}$, and four special-purpose *heads*. Heads may be thought of as pointers into the organism's genome and are similar to a traditional program counter or stack pointer. The instruction-head points to the next instruction to be executed. The flow-control head points to a location in the genome to which the instruction-head may be moved upon execution of certain instructions; it is similar to a `goto` label in the `C` programming language, but may be dynamically changed during execution. The read-head and write-head are used during replication, and enable the organism to read and write instructions within its genome.

Replication cycle

During the replication cycle an organism's genome experiences variation in the form of random instruction-level mutations. Figure 2 depicts the instructions comprising the replication cycle of the default AVIDA organism (mutations are likely to modify this sequence in descendants). The first step in replication is for the parent to allocate space for the offspring in its genome, and position the read- and write-heads at the beginning of its own genome, and its offspring's genome, respectively. The parent then executes its "copy-loop," where instructions are copied individually from the read-head to the write-head. Finally, the parent organism executes an `h-divide` instruction, which splits its genome into two parts, creating two organisms. Each time an instruction is copied, a mutation may be introduced according to a predefined probability. These mutations may take the form of a replacement (substituting a random instruction for the one copied), an insertion (inserting an additional, random instruction into the offspring's genome), or a deletion (removing the copied instruction from the offspring's genome).

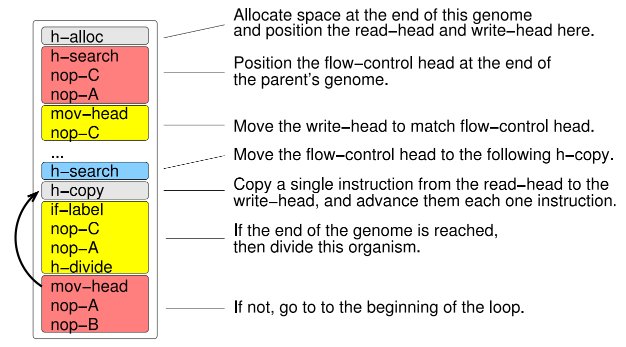

h–alloc
h–search
nop–C
nop–A
mov–head
nop–C
...
h–search
h–copy
if–label
nop–C
nop–A
h–divide
mov–head
nop–A
nop–B
Allocate space at the end of this genome and position the read–head and write–head here.
Position the flow–control head at the end of the parent's genome.
Move the write–head to match flow–control head.
Move the flow–control head to the following h–copy.
Copy a single instruction from the read–head to the write–head, and advance them each one instruction.
If the end of the genome is reached, then divide this organism.
If not, go to to the beginning of the loop.

Figure 2: Instruction sequence for the replication cycle of a typical AVIDA organism.

Merit

During an AVIDA experiment, the *merit* of a given digital organism determines how many instructions its virtual CPU is allowed to execute relative to the other organisms in the population, similar to a priority-based scheduling algorithm. For example, an organism with a merit of 2 will, on average, execute twice as many instructions as an organism with a merit of 1. Since digital organisms are self-replicating, a higher merit (all else being equal) results in an organism that replicates more frequently, spreading throughout and eventually dominating the population. Merit of a digital organism is updated based upon the *tasks* that are performed by the organism. Tasks are designed by the user and reward desirable behavior (they may also punish undesirable behavior), thereby driving natural selection. For example, in order to encourage communication within a population, a user might define a task that rewards an organism by doubling its merit when it sends a message to a neighboring organism. Tasks are generally defined in terms of externally visible behaviors of the organisms (their phenotype), rather than in terms of the specific instructions that must be executed by the digital organism's CPU. This approach allows maximum flexibility in the evolution of a solution for a particular task. The evolved solution might not be optimal when considering the task in isolation, but it is likely to have other properties that made it well-suited for its environment – robustness to mutation, for example.

2.2 Environment and Communication

Figure 3 depicts an AVIDA environment that has been subdivided into multiple demes (discussed further in Section 3). In the upper portion of this figure, we see that the environment comprises a number of *cells*, each of which can contain at most one organism; organisms cannot live outside of cells. Each cell has a circular list of directed *connections* to neighboring cells; these connections define the topology of the environment. The topology is configurable by the user. Currently, three topologies are available: GRID, TORUS, and CLIQUE (completely connected).

Each cell in the environment has a *facing*, a single connection selected from its connection list that defines the orientation of the resident organism. The facing of a cell may be used by the organism in a number of different ways. For example, an organism can send a message to the neighbor it is facing. The organism can also read and manipulate the facing of its cell via the `get-facing` and variants of the `rotate` instruction, respectively. Each cell has two associated identifiers, one that is allocated sequentially over all cells in the population, termed the *cell-sequential ID*, and another identifier that is a random 32-bit integer, termed the *cell-random ID*. A resident organism may obtain its cell identifier via the `get-id` instruction, which is configured by the user to refer to either the cell-sequential ID or the cell-random ID. These IDs might be used to identify the organism to its neighbors, for example by sending a message containing the ID.

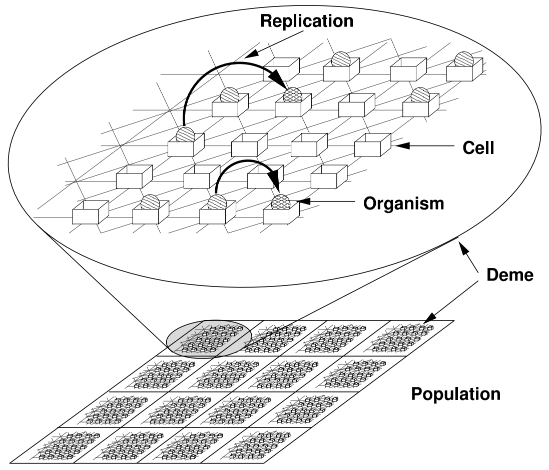


Figure 3: Depiction of an AVIDA environment of 16 demes, including cells and organisms within a single deme. An organism may replicate into a cell within its deme, replacing the resident organism (if present), while entire demes may be replaced during deme competition.

When an organism replicates, a target cell that will house the new organism is selected from the environment. Different models to select this target cell are available, including MASS-ACTION (select at random from among all cells) and NEIGHBORHOOD (select from cells adjacent to the parent), among others. In every case, an organism that is already present in the target cell is *replaced* (killed and overwritten) by the offspring. Figure 3 depicts the replication of two organisms.

Organisms in AVIDA can communicate with their neighbors using the `send-msg` and `retrieve-msg` instructions. Each message contains a *data* and a *label* field, both of which are 4-byte values. A third instruction, `if-inbox-empty`, returns a boolean value representing whether or not there are messages in the organism's inbox. Sending and receiving messages in AVIDA proceeds as follows. First, the sending organism must execute a `send-msg` instruction, which marshals two registers into a message and sends the message in the direction currently faced. If the sending organism is facing a neighboring organism, the message is deposited in that neighbor's inbox. If the sender was facing an empty cell, the message is lost. Finally, the recipient of the message must execute a `retrieve-msg` instruction to extract the message from its inbox and place its two fields into registers.

3. GROUP SELECTION IN AVIDA

The theory of group selection was originally proposed in 1962 by the biologist V. C. Wynne-Edwards [37], and later formalized into multilevel selection theory by D. S. Wilson and E. Sober [32, 36]. At a high level, multilevel selection theory states that groups of individuals may be vehicles for selection, similar to how a single individual is a vehicle for the selection of its components. In other words, multilevel selection posits that the survival of the individual is *linked* to the survival of the group. There are many different ways that these groups may be defined. For example, a group may be defined by a common trait (a trait-group), shared ancestry (clade selection), membership in the same species (species selection), or the interactions between related individuals (kin selection). Multilevel selection has been used in biology to explain the evolution of altruism, where an individual will sacrifice fitness for the benefit of the group, and the evolution of social behavior, particularly in populations of social insects, such as ant and bee colonies [3, 7, 29, 31].

In AVIDA, each deme in the population is configured with the same dimensions and environmental topology. Within each deme, organisms replicate, compete for resources, and are subject to mutations. When using demes, the cell in which an organism's offspring is placed belongs to the same deme as the parent, however, it is still selected according to the strategies described in Section 2.2. Entire demes periodically compete against each other based on a fitness function, where demes with a higher fitness probabilistically replace demes with a lower fitness. When a deme is replaced, all organisms in the target deme are removed, and each organism from the source deme is copied into the corresponding position in the target deme.

Deme competitions may be triggered periodically or as a reaction to a specific behavior occurring in the population. Deme competition may be thought of as an incomplete one-to-many mapping between D, the current set of demes in the population, and D', the resultant set of demes following the competition. First, the fitness of every deme in the population is calculated, and the summation of these fitnesses (T) is determined. For every deme d' in D', each deme d_i in D is replicated into d' with probability $P = \text{fitness}(d_i)/T$. This algorithm is similar in approach to a genetic algorithm, where the fitness function operates over a subpopulation of digital organisms.

The fitness of a deme in AVIDA is calculated by a user-defined fitness function. This fitness function takes as input the set of organisms in the deme, and produces a fitness value (a floating-point number) as output. Commonly used fitness functions include the number of replications, the average merit of organisms in the deme, and the average lifetime of organisms within the deme. Deme fitness functions can also be based on behavioral characteristics of organisms within the deme. For example, a fitness function may continually monitor each deme for a particular behavior (e.g., leader election) in order to increase their fitness during the next competition.

4. EXPERIMENTAL RESULTS

In this section we present experimental results into using AVIDA to evolve leader election in populations of digital organisms, where group selection is the only selective pressure acting upon the organisms. In related work [16], the authors have shown that leader election can evolve in a single AVIDA population, without group selection, but only under carefully designed environmental conditions. By using group selection instead of such an environment, we are able to decouple the desired cooperative behavior from the specific actions that must be performed by individual organisms within the population. The strength of this approach is that not specifying constituent behaviors enables evolution to discover novel strategies that might not otherwise have been apparent.

Experimental setup

In [16], we configured AVIDA to use a single population of 3,600 digital organisms in a 60×60 torus. For this study, we configured AVIDA with a population of 100 demes, each comprising 25 digital organisms. Each deme is configured in a 5×5 torus, except where noted. In both of these studies, experiments are run for 100,000 updates (an update averages 30 CPU instructions per organism, and is the standard unit of time in AVIDA). To account for the stochastic nature of evolution, 20 separate runs of AVIDA were performed for each experiment. Finally, the copy mutation rate was set to 0.75%, while the insertion and deletion mutation rates were set to 5.0%; these parameters correspond to the default AVIDA configuration.

Message-based leadership, single population

Let us first summarize the non-deme results [16]. Key to the success of evolving leader election in a single population was the environment in which the organisms lived. We defined the leader to be the organism with the numerically-largest ID, and we considered leader election to have occurred when greater than 95% of all messages sent carried this ID. Leader election evolved in the presence of three tasks, SEND-SELF, MAX-ID, and SEND-NON-ID. These first two tasks, SEND-SELF and MAX-ID, rewarded organisms for sending messages that carried their own ID and the maximum ID that had been received, respectively. The last task, SEND-NON-ID, penalized an organism for sending any message that did not carry an ID. We also showed that re-election of a leader is possible under this environment, however it requires that organisms voting for the old leader be selected *against* (via the SEND-NON-ID penalty).

Figure 4 depicts the typical messaging behavior of a population that evolved leader election using this environment. In this figure, four different values are plotted over the previous 100 updates: Total Sent, the total number of messages sent; ID-Carrying, the number of messages sent that carry any valid ID; Sender ID, the number of messages sent that carry the sender's ID; and >Sender ID, the number of messages sent that carry an ID greater than that of the sender. An indicator of successful leader election is that Total Sent, ID-carrying, and >Sender ID converged to approximately the same value. Here we see that each of these values converged to approximately 500,000 messages per 100 updates, while the number of Sender ID messages approached 0.

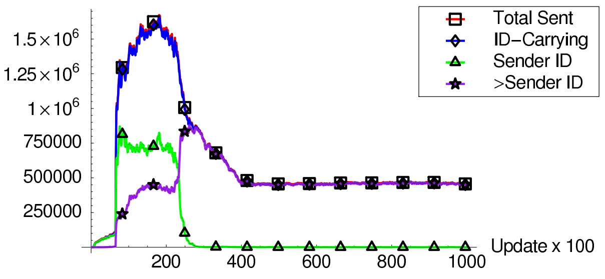


Figure 4: Typical messaging behavior for leader election in a single population.

Message-based leadership, without deme competition

In our first experiment using group selection, demes were not competed against each other, that is, no group-level selection was performed. This experiment was designed to determine if simply splitting the population into demes would have a substantial effect upon behavior. Figure 5 shows messaging behavior averaged over 20 different AVIDA experiments (runs). Here we see that of the 50,000

Total Sent messages, approximately 45,000 were Sender ID messages, thus indicating that leader election did not take place. This figure also shows that >Sender ID messages were not increasing during the run, implying that leader election would not occur even if the experiment were run longer.

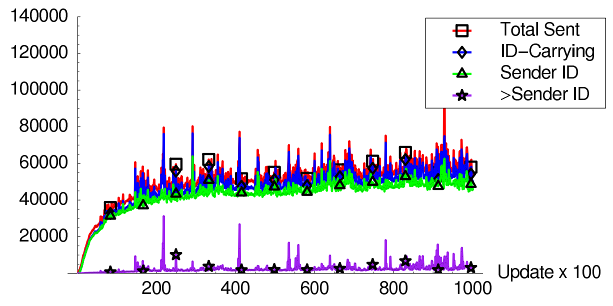


Figure 5: Messaging behavior using the SEND-SELF, MAX-ID, **and** SEND-NON-ID **tasks, with the population divided into demes, without deme competition.**

Message-based leadership, with deme competition

Our next experiment added competition between demes, where a fitness function, *MaxVote*, was used to probabilistically replicate demes according to the strategy described in Section 3. Two AVIDA instructions, SET-LEADER and GET-LEADER, were implemented to support *MaxVote*. These instructions set and retrieve the value of a variable internal to each organism that represents the leader of that organism, respectively; it may be thought of as a special-purpose register. The *MaxVote* fitness function has three primary components: S, the maximum size of the deme, in number of organisms; L, the number of leaders that have been elected by the deme since the previous deme competition; and C, the current maximum number of organisms in the deme that have called SET-LEADER with the same ID. Specifically, *MaxVote* $= (S*L+C)^2$, where in these experiments $S = 25$. Whenever at least 95% of the organisms in a single deme agree on the same leader, the leader's ID is reset and *MaxVote* records that an election has occurred (by incrementing L for that deme). For example, if at most 10 organisms in a deme agree on the same leader, the fitness of the deme will be 100: *MaxVote* $= (25*0+10)^2$. If, however, that same deme had already elected two leaders since the previous deme competition, its fitness will be 3,600: *MaxVote* $= (25*2+10)^2$. Demes compete with each other every 100 updates, thus *MaxVote* encourages each deme to elect as many leaders as possible over the course of 100 updates. Note that a fitness of 625 or greater indicates that a deme has elected at least one leader.

Figure 6 depicts maximum deme fitness averaged over 20 runs. We see here that the average deme was unable to elect an initial leader, since the fitness does not reach 625, though up to 20 out of 25 organisms did agree on a leader. Figure 7 shows the different message types sent, again averaged over 20 runs. Here we see that the number of >Sender ID messages are roughly 50% of all messages sent, indicating that organisms are forwarding IDs larger than their own.

Competition without tasks

Our next experiment removed the rewards for organisms that performed the MAX-ID, SEND-SELF, and SEND-NON-ID tasks. This experiment tests to see if group selection alone, without rewarding for constituent behaviors, is sufficient to evolve leader election. Figure 8 depicts the maximum deme fitness averaged over

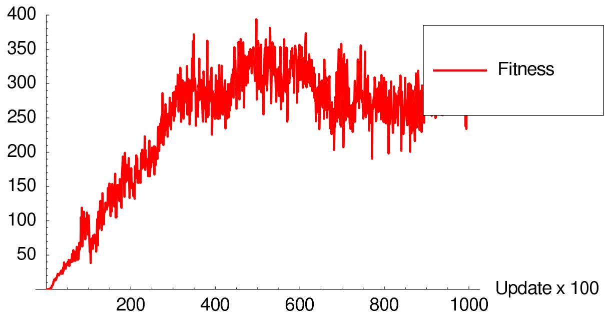


Figure 6: Maximum deme fitness using tasks and deme competition.

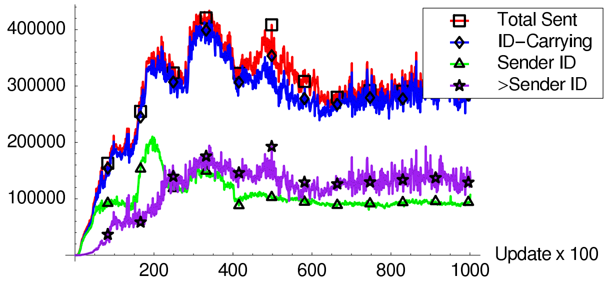


Figure 7: Messaging behavior using tasks and deme competition.

20 AVIDA runs without rewarding for tasks. Here we see that fitness is generally increasing, though it does not indicate that leader election has occurred. Figure 9 shows the message types averaged over 20 AVIDA runs. A notable difference between this experiment and those that include tasks is that 60% of messages do not carry an ID. However of those messages that carry an ID, 50% carry an ID larger than the sender's, indicating that the same leader election strategy is being employed.

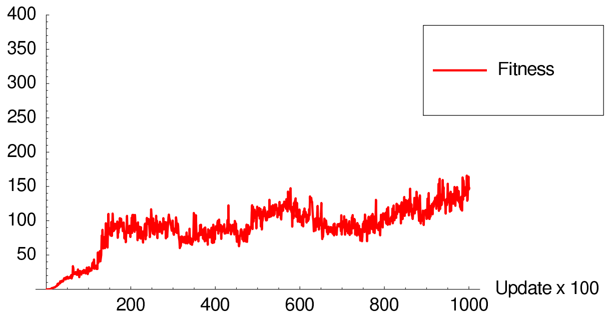


Figure 8: Maximum deme fitness without using tasks, with deme competition.

Competition in a clique

In the experiments described to this point, organisms have no direct mechanism to examine their neighbors. In this experiment, we add another instruction to AVIDA, GET-NEIGHBOR-ID, that directly returns the ID of the faced organism. We also change the topology of each deme to a clique, so that each organism is capable of calling GET-NEIGHBOR-ID on every other organism in the deme. We note that the introduction of GET-NEIGHBOR-ID, without also altering the topology, had little effect on experimental results.

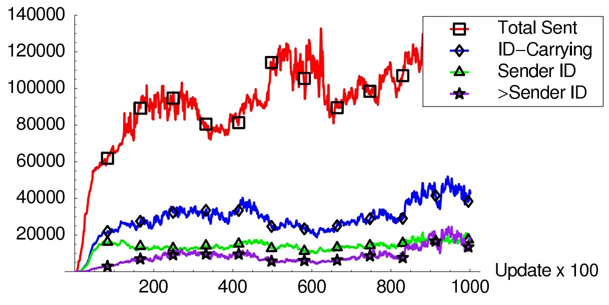


Figure 9: Messaging behavior without using tasks, with deme competition.

Figure 10 depicts the maximum deme fitness averaged over 20 AVIDA runs. Here we see that fitness is steadily increasing, and is an improvement over the average fitness shown in Figure 8. However, as in the previous experiment, no evidence of leader election is found. Figure 11 shows the message types averaged over these same 20 AVIDA runs. Here we see that messages are now almost completely non-ID carrying, indicating that other means (e.g., the GET-NEIGHBOR-ID instruction) are being used to retrieve IDs. This experiment indicates that topology is an important factor for the evolution of leader election.

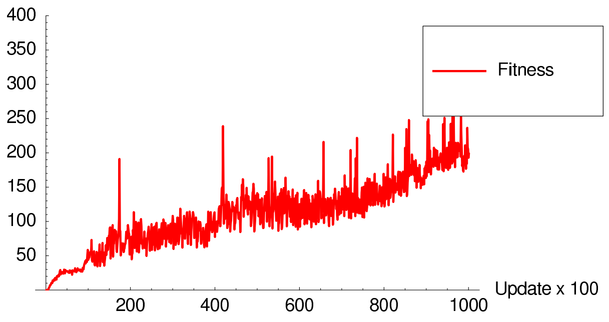


Figure 10: Maximum deme fitness without using tasks, with deme competition, in a clique topology.

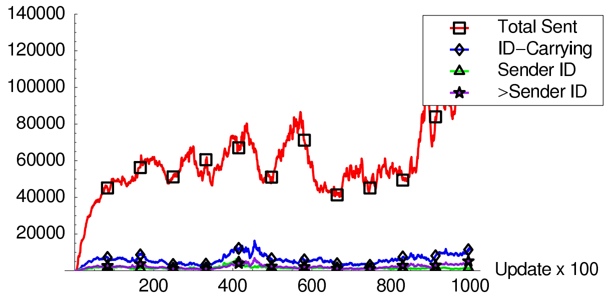
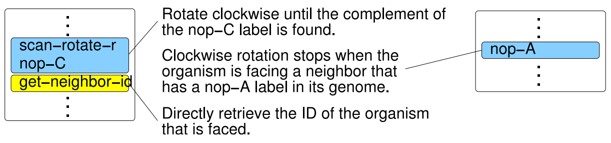


Figure 11: Messaging behavior without using tasks, with deme competition, in a clique topology.

Competition in a clique, with neighbor scanning

Using the ROTATE family of instructions, organisms in AVIDA have the capability to rotate through their list of neighbors, where the particular neighbors of an organism are dependent upon the topology. There are a number of variations of the rotate instruction. One such variant is SCAN-ROTATE-{L,R}, where the organism will rotate to the first neighbor that contains a specified *label*. A label in AVIDA is a series of one or more NOP-{A,B,C} instructions anywhere in the organism's genome. When a SCAN-ROTATE-{L,R} instruction is followed by a label, it will rotate left (counter-clockwise) or right (clockwise) until it faces a neighbor whose genome contains the *complement* label. Complements are formed from the following replacements: NOP-A→NOP-B; NOP-B→NOP-C; NOP-C→NOP-A. In every case, rotation stops at the first neighbor containing the complement, or at the original facing, whichever comes first. An example of the SCAN-ROTATE-R instruction and the use of labels is shown in Figure 12. This figure contains fragments of two genomes, where the three instructions in the left-most genome scan neighboring organisms for the label shown in the right-most genome. Once found, the GET-NEIGHBOR-ID instruction will place the ID of the faced neighbor directly into a register.



scan–rotate–r
nop–C
get–neighbor–id
Rotate clockwise until the complement of the nop–C label is found.
Clockwise rotation stops when the organism is facing a neighbor that has a nop–A label in its genome.
Directly retrieve the ID of the organism that is faced.
nop–A

Figure 12: An example of the usage of labels in AVIDA.

In this experiment, we configured AVIDA to use the SCAN-ROTATE-{L,R} instructions, while keeping the topology a clique. Figure 13 depicts the maximum deme fitness averaged over 20 AVIDA runs (please note the scale change). Here we see a significant improvement over previous experiments, with the average deme electing just under 3 leaders in 100 updates. Figure 14 depicts the messaging behavior averaged over these same 20 AVIDA runs, where we see that again, the majority of messages do not carry an ID. So, while organisms are successfully electing leaders, they are apparently not using messages to do so.

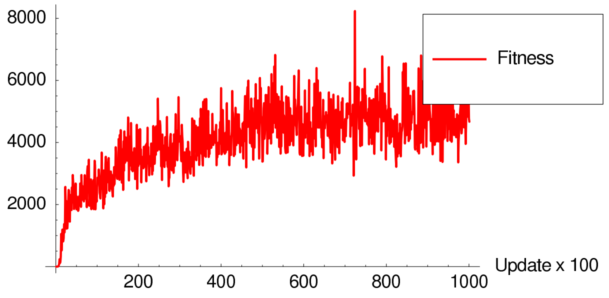


Figure 13: Maximum deme fitness without using tasks, with deme competition, in a clique topology, with neighbor scanning.

We investigated this result by examining the best-performing deme out of all AVIDA runs, which elected 9 leaders during a 100 update period. Figure 15 shows the fitness of that deme, which reaches a maximum of 51,076 at update 78,900.

Figure 16 shows the 12 different genomes present in this deme during the election of a leader (13 of the 25 organisms in this deme shared a genome). Each genome contains a common sequence of instructions that implements leader election, shown at the top of this figure. Specifically, each organism rotates clockwise, stopping at the first organism that contains a `nop−A` label (the complement of the `nop−C` label). It then sets its leader ID to the ID of the faced organism. Every time this deme elected a leader, there was exactly

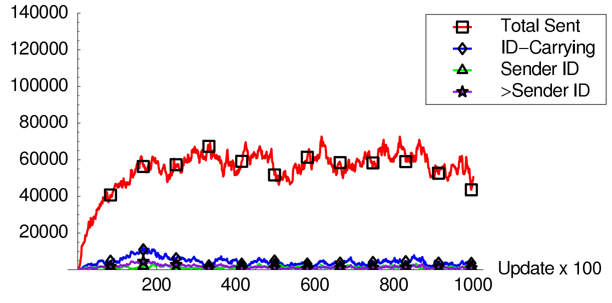


Figure 14: Messaging behavior without using tasks, with deme competition, in a clique topology, with neighbor scanning.

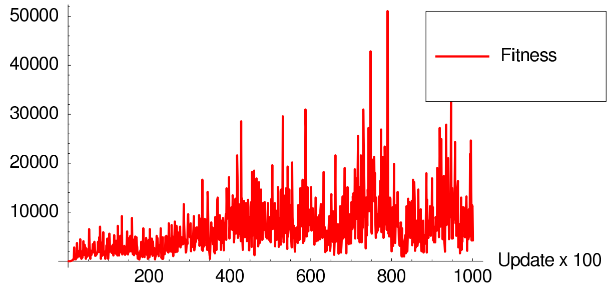


Figure 15: Maximum fitness of the best-performing population using deme competition, in a clique topology, using neighbor scanning. Maximum fitness of 51,076 occurs at update 78,900.

one organism in the deme that contained a `nop-A` label, shown at the bottom of this figure. In other words, a leader was elected whenever exactly one organism contained a specific identifying feature that the group had agreed upon. Interestingly, this approach is similar to the immunological question of detecting "non-self," where an immune system will attack cells that are identified as not belonging to the host [5].

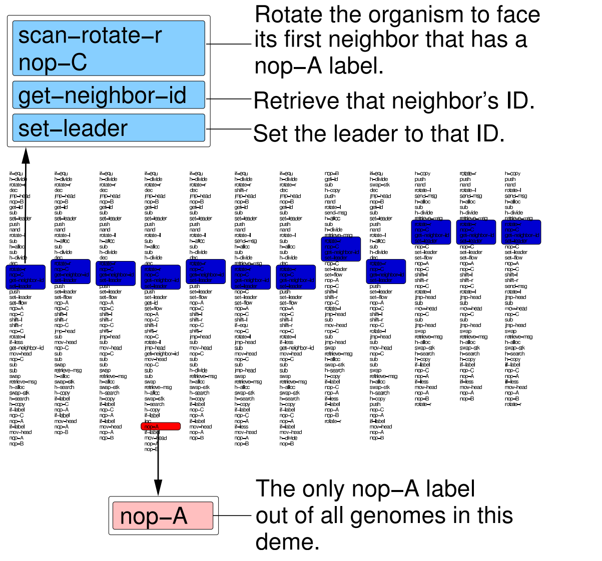
5. CONCLUSIONS AND FUTURE WORK

We have demonstrated that digital evolution, in combination with group selection, can produce a relatively complex cooperative behavior, leader election, in a population of digital organisms. Furthermore, we have shown that in the presence of mutations, populations of organisms are able to recover from the death of the leader multiple times. Moreover, digital evolution and group selection can produce these behaviors by selecting for the global *effect* of leader election, rather than by specifying tasks or roles for individual organisms.

Our ongoing and future investigations include the following. First, we are using AVIDA to study the evolution of other distributed operations, such as data gathering, which can be applied to wireless sensor networks. Second, to study evolution in individuals capable of movement, such as mobile robotic agents, 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.

Further Information.

Technical papers on digital evolution, along with downloads of the AVIDA software are available at the Digital Evolution Laboratory website: `http://devolab.cse.msu.edu`. Related publications of the Software Engineering and Network Systems Laboratory are available at `http://www.cse.msu.edu/sens`.



scan−rotate−r
nop−C
get−neighbor−id
set−leader
Rotate the organism to face its first neighbor that has a nop−A label.
Retrieve that neighbor's ID.
Set the leader to that ID.
nop−A
The only nop−A label out of all genomes in this deme.

Figure 16: Depiction of the 12 different genomes active in the deme that elected 9 leaders. 13 organisms shared a genome, while exactly one organism had a genome containing the NOP-A instruction, which indicated the leader.

Acknowledgments.

The authors gratefully acknowledge the contributions of the following individuals to this work: Ben Beckmann, Jeffrey Clune, Sherri Goings, David Bryson, Richard Lenski, Heather Goldsby, and Betty Cheng. 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, U.S. National Science Foundation grants EIA-0130724, ITR-0313142, and CCF 0523449, and a Quality Fund Concept grant from Michigan State University.

6. REFERENCES

[1] C. Adami, C. Ofria, and T. C. Collier. Evolution of biological complexity. *Proceedings of the National Academy of Sciences*, 97:4463–4468, 2000.

[2] 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*, 1(1):26–66, 2006.

[3] A. F. G. Bourke and N. R. Franks. *Social Evolution in Ants*. Princeton University Press, 1995.

[4] J. Branke. *Evolutionary Optimization in Dynamic Environments*, volume 3 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[5] E. Cohen. Figuring immunity: Towards the genealogy of a metaphor. In A.-M. Moulin and A. Cambrosio, editors, *Singular Selves: Historical Issues and Contemporary Debates in Immunology*, pages 179–201. Elsevier, Amsterdam, 2001.

[6] P. Dasgupta. Intelligent agent enabled genetic ant algorithm for P2P resource discovery. In *Third International Workshop*

on Agents and Peer-to-Peer Computing, pages 213–220, 2004.

[7] M. J. W. Eberhard. The evolution of social behavior by kin selection. *The Quarterly Review of Biology*, 50(1):1–33, March 1975.

[8] O. Giel and P. K. Lehre. On the effect of populations in evolutionary multi-objective optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 651–658, New York, NY, USA, 2006. ACM Press.

[9] S. Goings, J. Clune, C. Ofria, and R. T. Pennock. Kin selection: The rise and fall of kin-cheaters. In *Proceedings of the Ninth International Conference on Artificial Life*, pages 303–308, Boston, MA, USA, September 2004.

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

[11] M. Guimaraes and L. Rodrigues. A genetic algorithm for multicast mapping in publish-subscribe systems. In *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 67– 74, April 2003.

[12] A. Hara and T. Nagao. Automatically defined groups for multi-agent cooperation. In *Proceedings of the 2nd Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*, pages 91–98, 1998.

[13] 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.

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

[15] J. O. Kephart and D. M. Chess. The vision of autonomic computing. In *IEEE Computer*, volume 36, pages 41–50, 2003.

[16] D. B. Knoester, P. K. McKinley, B. Beckmann, and C. A. Ofria. Evolution of leader election in populations of self-replicating digital organisms. Technical Report MSU-CSE-06-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2006.

[17] 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 edition, 2005.

[18] H. Kwong and C. Jacob. Evolutionary exploration of dynamic swarm behaviour. In *Congress on Evolutionary Computation (CEC'2003)*, pages 367–374, Canberra, Australia, December 2003. IEEE.

[19] R. E. Lenski, C. Ofria, T. C. Collier, and C. Adami. Genome complexity, robustness, and genetic interactions in digital organisms. In *Nature*, volume 400, pages 661–664, 1999.

[20] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami. The evolutionary origin of complex features. In *Nature*, volume 423, pages 139–144, 2003.

[21] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[22] D. Marocco, A. Cangelosi, and S. Nolfi. The role of social and cognitive factors in the emergence of communication: Experiments in evolutionary robotics. In *Philosophical Transactions of the Royal Society London – A*, volume 361, pages 2397–2421, 2003.

[23] A. Martinoli and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In *Proceedings of the Fourth Symposium on Experimental Robotics ISER-95*, June 1995.

[24] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *IEEE Computer*, 37(7):56–64, 2004.

[25] T. Murata and T. Nakamura. Genetic network programming with automatically defined groups for assigning proper roles to multiple agents. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pages 1705–1712, New York, NY, USA, 2005. ACM Press.

[26] C. Ofria and C. Adami. Evolution of genetic organization in digital organisms. In *Proceedings of DIMACS Workshop on Evolution as Computation*, pages 167–175, Princeton, NJ, 1999.

[27] C. Ofria, C. Adami, and T. Collier. Design of evolvable computer languages. In *IEEE Transactions in Evolutionary Computation*, volume 17, pages 528–532, 2002.

[28] C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. In *Journal of Artificial Life*, volume 10, pages 191–229. International Society of Artificial Life (ISAL), March 2004.

[29] D. C. Queller and J. E. Strassmann. Kin selection and social insects. *BioScience*, 48(3):165–175, March 1998.

[30] R. Schoonderwoerd, J. L. Bruten, O. E. Holland, and L. J. M. Rothkrantz. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2004.

[31] T. D. Seeley. Honey bee colonies are group-level adaptive units. *The American Naturalist*, 150:S22–S41, July 1997.

[32] E. Sober and D. S. Wilson. *Unto Others: The Evolution and Psychology of Unselfish Behavior*. Harvard University Press, 1998.

[33] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, 2005.

[34] C. O. Wilke, J. L. Wang, C. Ofria, C. Adami, and R. E. Lenski. Evolution of digital organisms at high mutation rate leads to survival of the flattest. In *Nature*, volume 412, pages 331–333, 2001.

[35] D. S. Wilson. A theory of group selection. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 72, pages 143–146, January 1975.

[36] D. S. Wilson. Altruism and organism: Disentangling the themes of multilevel selection theory. *The American Naturalist*, 150:S122–S134, July 1997.

[37] V. Wynne-Edwards. *Animal Dispersion in Relation to Social Behavior*. Oliver & Boyd, London, 1962.

[38] L. Yamamoto and C. F. Tschudin. Experiments on the automatic evolution of protocols using genetic programming. In I. Stavrakakis and M. Smirnov, editors, *Autonomic Communication, Second International IFIP Workshop, WAC 2005, Revised Selected Papers*, volume 3854 of *Lecture Notes in Computer Science*, pages 13–28, Athens, Greece, Oct.2-5 2005. Springer.

[39] E. M. Zechman and S. R. Ranjithan. Multipopulation cooperative coevolutionary programming (mccp) to enhance design innovation. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, pages 1641–1648, New York, NY, USA, 2005. ACM Press.