

Evolution of Probabilistic Consensus in Digital Organisms

David B. Knoester and Philip K. McKinley
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
Email: {dk, mckinley}@cse.msu.edu

Abstract—The complexity of distributed computing systems and their increasing interaction with the physical world impose challenging requirements in terms of adaptation, robustness, and resilience to attack. Based on their reliance on heuristics, algorithms for consensus, where members of a group agree on a course of action, are particularly sensitive to these conditions. Given the ability of natural organisms to respond to adversity, many researchers have investigated biologically-inspired approaches to designing robust distributed systems. In this paper, we describe a study in the use of digital evolution, a type of artificial life system, to produce a distributed behavior for reaching consensus. The evolved algorithm employs a novel mechanism for probabilistically reaching consensus based on the frequency of messaging. Moreover, this design approach enables us to change parameters based on the specifics of the desired system, with evolution producing corresponding flavors of consensus algorithms. Our results demonstrate that artificial life systems can be used to discover solutions to engineering problems, and that experiments in artificial life can inspire new studies in distributed protocol development.

Keywords-distributed algorithm, consensus, evolutionary computation, digital evolution, self-organization.

I. INTRODUCTION

There are many examples of organisms in nature that exhibit cooperative behaviors of varying complexity. Some of these cooperative behaviors include *consensus*, where members of a group agree upon a particular course of action. Consensus behaviors are visible throughout the spectrum of life, from decision making in humans [1], to leader election in schools of stickleback fish [2], to quorum sensing in bacteria [3]. Improving our understanding of such behaviors has the potential to advance not only the biological sciences, but also to aid in the design of computational systems.

In distributed computing systems, consensus algorithms are frequently used to ensure consistency between replicated components in an effort to increase reliability [4] or to provide the basis for distributed lock management [5]. However, because bounded-time consensus in the presence of failures has been proven impossible (the so-called FLP impossibility proof) [6], numerous heuristics for consensus have been developed [7]. As computing systems continue to expand their reach into the natural world, for example through cyber-physical systems, as well as scaling up in size and complexity, designers are faced with a multitude

of additional complications, some of which may no longer be amenable to traditional algorithms.

Many of the challenges faced by designers of distributed systems are shared by biological organisms. For example, node churn, message loss, and network segmentation all have biological analogues in organism death, sensory ineffectiveness, and environmental dangers, respectively. As such, many recent approaches to designing distributed systems have focused on *biomimetics*, where behaviors observed in nature are replicated *in silico* [8]. A complementary approach to biomimetics is evolutionary computation, where instead of mimicking the behaviors found in nature, we harness the power of evolution and natural selection, the processes that produced those behaviors. Compared to traditional approaches for designing distributed algorithms, evolutionary computation enables researchers to specify the desired global behaviors of the system directly and without *a priori* knowledge of required local behavior. Instead, these local behaviors *evolve* in response to selective pressures derived from the specified objective. Moreover, while still relatively early in its application to distributed systems, evolutionary computation enables the designer to search an enormous solution space, often revealing robust and non-intuitive solutions.

In this study, we use *digital evolution* [9], a form of evolutionary computation, to evolve digital organisms that exhibit consensus. In digital evolution, a population of digital organisms (self-replicating computer programs) exists in a user-defined computational environment. These organisms replicate, compete for resources, and are subject to mutation and natural selection. Over thousands of generations, they can evolve to survive, and even thrive, under extremely dynamic and adverse conditions. In this study, we employ AVIDA [10], a digital evolution platform previously used to study biological evolution. Digital organisms in AVIDA exist in a spatial environment, communicate with their neighbors, and execute their “genome,” a list of virtual CPU instructions. Recently, AVIDA has been applied to an increasingly diverse set of engineering problems, from the construction of communication networks [11] to adaptive population control [12].

The contributions of this work are as follows: First, we were able to evolve a genome that exhibits consensus when

placed within a group comprising multiple copies of itself. Second, by analyzing this genome, we show that digital evolution produced a novel form of consensus based on probabilistic message forwarding. Third, we simulated this algorithm to compare it to examples of distributed consensus from the literature. These results indicate that evolutionary computation in general, and digital evolution in particular, offer a promising approach to designing distributed algorithms.

II. RELATED WORK

Replication of the critical components of a distributed computing system is a well-known method to improve overall system reliability [13]. Consensus algorithms are frequently used to ensure that state information shared among these replica remains consistent [4], [7]. In practice, algorithms such as Paxos [14] are used as the basis for implementations of consensus [15], which in turn can support higher-level services such as distributed lock management [5]. As distributed systems continue to increase in scale and complexity, new approaches to the design and implementation of such algorithms are needed, particularly as the heuristics used for fault-tolerance are overtaken by the complexity of the environments in which these systems are deployed. Indeed, numerous distributed algorithms already integrate techniques from dynamic systems [16] to mitigate the effects of these environments.

The consensus problem has been approached from many fields of research. While the motivation for this paper stems from consensus in distributed systems, consensus is studied in fields as diverse as coordination games [17] and cooperative control [18]. Coordinating the behavior of multiple agents is also a common problem in evolutionary robotics [19], and artificial life studies have contributed to our understanding of the evolution of cooperation and communication [20].

In this study, we use a model of the consensus problem based on [7] and [21], called the *n-process id consensus problem*, in which a group of processes seek to reach agreement, or consensus, upon a common process id. The *n-process id consensus problem* is similar to the alignment problem [22], where the set of possible states is simply the *n* process ids. Necessarily, solving the consensus problem requires cooperation and communication among the processes, in addition to a strategy for selecting the agreed-upon value. Additional related forms of consensus will be discussed in Section VII. Given the fundamental usefulness of consensus in distributed algorithms, as well as its theoretical impossibility [6], discovering heuristics for achieving consensus in modern distributed systems is of great importance.

Previous studies that have used evolutionary algorithms to produce cooperative behaviors have tended to focus on either the evolution of cooperation under fixed communication

properties [23], or the evolution of communication under fixed cooperation properties [24]. Here, we provide digital organisms with a mechanism for communication and select for the ultimate (as opposed to proximal) result of cooperation, leaving evolution to discover the specific cooperative and communication behaviors.

III. DIGITAL EVOLUTION AND AVIDA

Digital evolution [9] is a form of evolutionary computation originally developed to study evolution in biology. AVIDA [10], a platform for digital evolution, has also recently been applied to engineering problems, including the construction of communication networks [11] and adaptive population control for energy efficiency [12]. Although AVIDA has many characteristics that make it suitable for studying evolution in biology, here we use AVIDA similarly to a linear genetic program [25].

There are a number of features that make AVIDA an appropriate choice for evolving distributed algorithms, and these features have already made it possible for us to study various types of cooperative behaviors. First, digital organisms in AVIDA have only rudimentary computation capabilities, comparable to resource-constrained nodes in a sensor network. Second, though organisms in AVIDA live in a digital world that enables them to communicate with each other, any communication behaviors must evolve within an environment exhibiting various communication hazards. Third, AVIDA includes features that enable the evolution of group behaviors. In the remainder of this section, we briefly describe the structure of a digital organism in AVIDA and the mechanism by which group behaviors can be evolved.

A. 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 exists in a common virtual environment. Within this 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 to a traditional assembly language. Instructions enable an organism to perform simple mathematical operations, control execution flow, communicate with neighboring organisms, and replicate. The virtual CPU architecture used in this study contains a circular list of three general-purpose registers $\{AX, BX, CX\}$ and two general-purpose stacks $\{GS, LS\}$.

As shown in Figure 1, each organism in AVIDA lives in a *cell* located in a fixed location within a spatial environment. Each cell can contain at most one organism; organisms cannot live outside of cells. The topology of the environment defines the neighborhood of each cell, and is user-defined. For example, the environment topology may be configured as a grid, a torus, or as a well-mixed environment, where

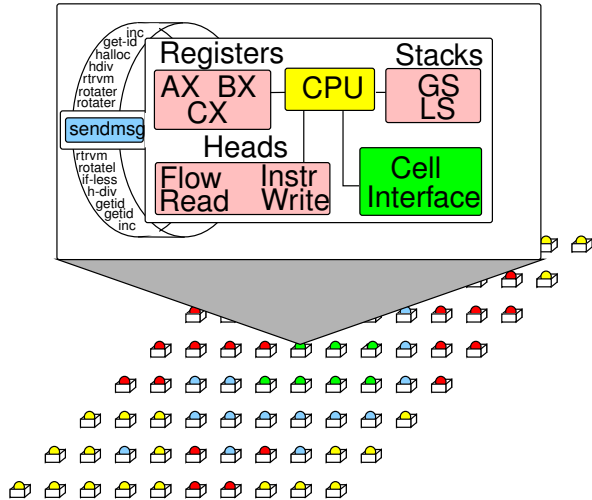


Figure 1. An Avida population containing multiple genomes (bottom), and the structure of an individual organism (top).

all cells are neighbors of each other (also referred to as a clique). Furthermore, each organism in the environment has a *facing* that defines its orientation. This facing may be used in a number of different ways. For example, an organism can send a message in the faced direction. The organism can also sense and manipulate its facing via the `get-facing` and `rotate-*` instructions, respectively.

B. Levels of Selection

Figure 2 depicts the three different levels of selection available within AVIDA. Under the first, *individual selection*, organisms compete with each other for space (cells) in their environment and are responsible for their own replication, that is, organisms must execute instructions to self-replicate. In the second level, *group selection*, the population of digital organisms is divided into distinct subpopulations, called *demes*. Within each deme, organisms replicate, mutate, and compete with each other for space and resources. At the same time, demes also compete with each other for space and resources based on the behavior of their constituent organisms. The third level of selection available within AVIDA is most similar to *multicellularity* in biology. Here, the population is again split into demes, however, the organisms within each deme are homogeneous. In this case, a genome is attached to each deme, rather than individual organisms, and all organisms within the deme are instantiations of that same genome. When a deme replicates, any mutations occur to the deme’s genome, which is called a *digital germline* [11].

For this study, we used *CompeteDemes*, a framework within AVIDA that enables the periodic replication and competition of demes, in combination with a digital germline to ensure homogeneity within demes. During the execution of an AVIDA trial, the *CompeteDemes* framework period-

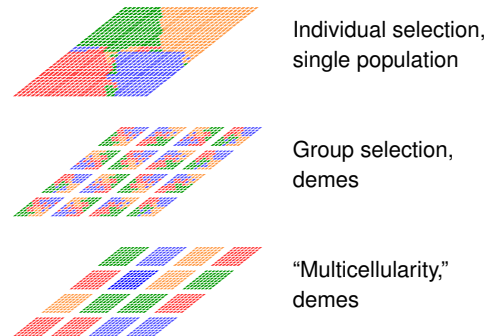


Figure 2. Levels of selection available within Avida; different shades represent different genomes.

ically 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, and replacing those demes with lower fitness. For this study, we define fitness functions based on the degree to which organisms within a deme achieve consensus. Over time, the *CompeteDemes* framework will preferentially replicate those demes that are more capable of reaching consensus than others, resulting in a population that evolves increasingly better solutions. At the end of an AVIDA trial, the dominant, or most prolific, genome can be identified for further analysis of its behavior.

IV. METHODS

Typically, the AVIDA user will configure the environment in which the digital organisms live and define the selective pressures that act upon the population. Once configured, multiple AVIDA trials are conducted to account for the stochastic nature of evolution; although a single result may be sufficient when searching for an effective algorithm, multiple trials improve statistical accuracy. In the remainder of this section, we describe the configurations and extensions to AVIDA that were required for this study of the evolution of consensus.

Instructions: All relevant instructions employed in this study are summarized in Table I. Of particular note are the instructions associated with messaging, “opinions,” and the “flash” capability. These instructions enable organisms to send and retrieve messages; manipulate their opinion register, used as the basis for the fitness functions that will be presented in Section V; and to both sense and trigger virtual flashes, a synchronization primitive based on the behavior of fireflies, respectively.

Configuration: We configured AVIDA to use the *CompeteDemes* process, outlined in Section III-B, with specific configuration values summarized in Table II. We

Table I
RELEVANT INSTRUCTIONS FOR THIS STUDY. ALL INSTRUCTIONS ARE
EQUALLY LIKELY TO BE SELECTED AS TARGETS FOR MUTATION.

Instruction	Description
send-msg	Sends a message to the neighbor currently faced by the caller; message contains contents of <i>BX</i> and <i>CX</i> registers.
retrieve-msg	Loads the caller's <i>BX</i> and <i>CX</i> registers from a previously received message.
rotate-left-one	Rotates the caller counter-clockwise one step.
rotate-right-one	Rotates the caller clockwise one step.
get-opinion	Sets register <i>BX</i> to the value of the caller's opinion register.
set-opinion	Sets the caller's opinion register to the value in register <i>BX</i> .
bcast1	Sends a two-word message containing the values of registers <i>BX</i> and <i>CX</i> to all immediately neighboring organisms.
collect-cell-data	Sets register <i>BX</i> to the value of the cell data where the caller lives.
get-neighborhood	Load a hidden register with a list of the IDs of all neighboring organisms.
if-neighborhood-changed	Execute the subsequent instruction if the caller's current neighbor is different from that when <i>get-neighborhood</i> was last called.
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, then execute the subsequent instruction. Otherwise, skip the subsequent instruction.
flash-info	If the caller has ever received a flash, then set <i>BX</i> to 1 and <i>CX</i> to the number of cycles since that flash was received. Otherwise, set <i>BX</i> and <i>CX</i> to 0.

used 400 demes, each comprising 25 digital organisms connected in a torus topology. Each deme was configured to use a germline to provide homogeneity within demes. Each time a deme replicated, the offspring deme was filled with 25 copies of the latest (possibly mutated) genome from that deme's germline.

Table II
COMMON AVIDA CONFIGURATIONS USED FOR THE EXPERIMENTS
DESCRIBED IN THIS STUDY.

Configuration	Value
Trials per experiment	30
Max. population size	10,000
Number of demes	400, each 5×5
Environment topology	Torus
Copy mutation rate	0.0075 (per instruction)
Insertion mutation rate	0.05 (per replication)
Deletion mutation rate	0.05 (per replication)
Time slice	5 instructions per update
CompeteDemes	Compete all demes every 800 updates

V. RESULTS

A. Simple Consensus

Our initial experiment in the evolution of consensus focused on the Simple Consensus Dilemma (SCD), based on the n -process id consensus problem. Informally, we define the SCD as follows¹:

Each agent in a group is assigned a unique identifier. Agents within this group are independent, and can communicate by sending messages. Each agent can also designate a value, selected from the set of identifiers, as its "opinion." Following a period of time during which the agents may communicate with each other, the opinion of every agent within the group is examined. If all agents express the same opinion, the group survives. If agents express different opinions, the group perishes. How should the agents act?

The question we are asking then, is: can evolution solve the SCD, and if so, what strategies will be employed? We configured each deme as a 5×5 torus of cells, where each of these cells was assigned a random 32-bit integer as *cell data*, which we used as the identifier for the organism in that cell. We then defined a fitness function to reward demes whose constituent organisms set their opinion to a common cell data. The specific fitness function used here was:

$$F = (1 + S_{max})^2 \quad (1)$$

where F is the resulting fitness value and S_{max} is the *maximum support*, or the number of organisms that have expressed the most common opinion. We emphasize that although organisms are able to set their opinion to any integer value, only opinions that are set to cell data present within the deme contribute to the deme's fitness.

Figure 3 plots the average and best-case performance of individual demes across all 30 trials. In this figure, the y -axis represents the fraction of consensus achieved, where a value of 1 is complete consensus, and the x -axis is in units of *updates*, the standard unit of time in AVIDA; an update corresponds to each organism receiving, on average, five virtual CPU cycles. Here we see that the best-performing demes achieve consensus after approximately 25,000 updates, while the average deme steadily approaches consensus.

At the end of 30 trials, six of the dominant (most prolific) genomes appeared to employ a strategy that searched for the maximum cell data, while an additional five genomes searched for the minimum cell data. These approaches are similar to an evolved strategy from an earlier study on the diffusion of the maximal sensed value [26]. In that study, we used AVIDA *tasks*, a mechanism to apply selective pressure

¹This presentation of the consensus problem is intentionally patterned after the classic Prisoner's Dilemma game. Game theoretic studies have historically provided insight into many of the fundamental aspects of evolution, such as cooperation [23].

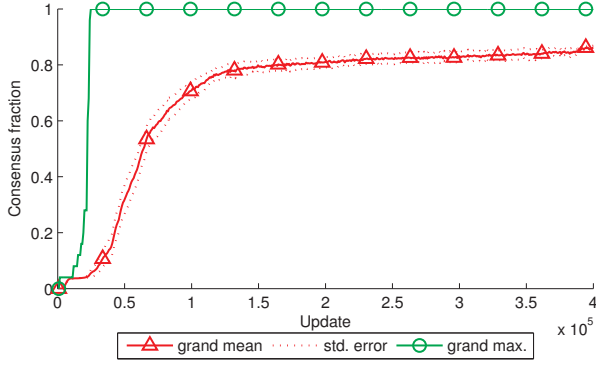


Figure 3. Deme performance for the Simple Consensus Dilemma.

to individuals, in order to encourage the evolution of a distributed behavior that searched for the maximum cell data. Here, however, we employed a fitness function that operated only at the group level, enabling evolution to discover the most fit strategy without any guidance as to how consensus should be reached.

Figure 4 depicts a fragment of an evolved genome that solves the SCD by searching for the maximum cell data. At the individual level, this genome causes the organism to set its opinion to either the value of data contained in its own cell, or the contents of a received message, whichever is larger. When a group of organisms all share this genome, each will eventually set its opinion to the maximum cell data to which any of the group members have access.

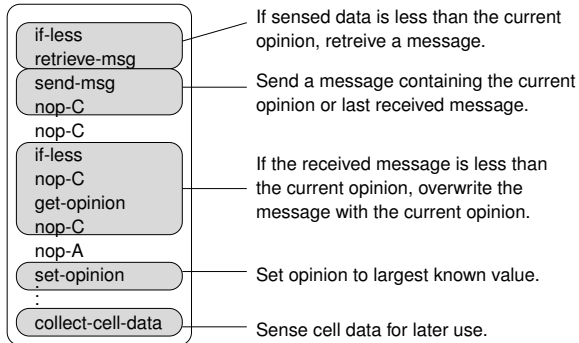


Figure 4. Genome fragment responsible for searching for the maximum cell data present within a deme.

B. Iterated Consensus

In the next series of experiments, we explored the *iterated consensus dilemma*. The Iterated Consensus Dilemma (ICD) modifies the SCD by introducing *rounds*, where the group of agents are repeatedly tasked with reaching consensus. Whenever a group reaches consensus, the agent with the identifier that the group has agreed upon is replaced by a new agent with a different identifier. The particular issue being examined here is whether a group of organisms can

reach consensus, sense that the value that was agreed upon is no longer valid, and then recover to reach consensus on a different value. ICD may be thought of as non-stationary optimization, where the group must continually strive to reach consensus in the face of population turnover. To study the evolution of behaviors that solve the ICD, we modified the fitness function described in Section V-A. The specific fitness function we used here was:

$$F = (1 + S_{max} + R \cdot D)^2 \quad (2)$$

where F is the resulting fitness, S_{max} is the maximum support, R is the number of times the deme has reached consensus during this competition period, and D is the size of the deme (always 25 in this experiment). To determine R , each deme in the population is evaluated at every update to determine if its constituents have reached consensus. If so, the value of R for that deme is incremented. Whenever consensus is reached, the cell data corresponding to the agreed-upon value is reset to a random integer from the range bounded by the maximum and minimum cell data present within that deme, and the organism in that cell is replaced (killed and overwritten) with a new organism instantiated from the germline.

Figure 5 plots the average and best-case performance of individual demes across all 30 trials. The data show that the average deme approaches a single consensus round per competition period, while the best-case performance of any deme is two consensus rounds. The remaining experiments examine various ways in which evolution might discover improved solutions to the ICD.

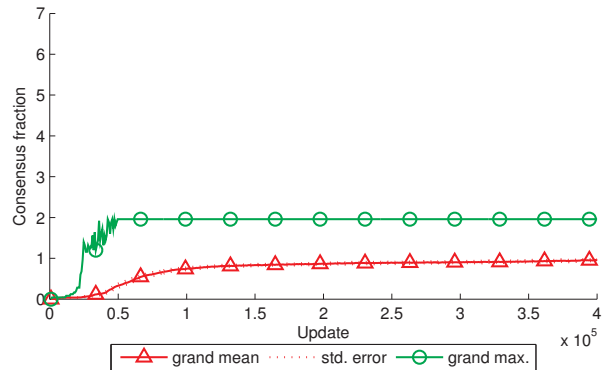


Figure 5. Deme performance for the Iterated Consensus Dilemma.

C. Neighborhood Sensing

In this experiment, we provide additional instructions that organisms may use to sense their neighborhood for changes. Specifically, we add the `get-neighborhood` and `if-neighborhood-changed` instructions, which enable individuals to sense their environment and determine if the organisms in their neighborhood are different than the last

time the neighborhood was sensed. This capability could, for example, be used to sense the replacement of an organism once consensus had been reached, and thus trigger the beginning of a new consensus round.

Figure 6 plots average and best-case performance of individual demes across all 30 trials. Although the mean behavior of demes from this experiment is significantly different from that in Section V-B ($p < 0.001$, Mann-Whitney U -test), the best-case performance of any deme across all trials is unchanged. Based on these results, we conclude that the ability of an individual to sense their neighborhood does not aid in the evolution of consensus over organisms without this ability.

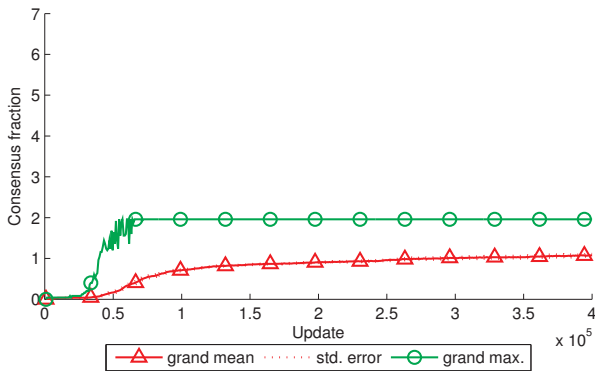


Figure 6. Deme performance on the Iterated Consensus Dilemma, with the addition of neighborhood-sensing instructions.

D. Broadcast

All previous experiments have used relatively simple mechanisms for sending and receiving messages. Specifically, individuals in the previous experiments have only been able to send point-to-point messages. Here, we present a treatment that improved the message-sending capability of digital organisms by adding the `bcast1` instruction; this instruction functions identically to `send-msg` in terms of register usage and payload, however, the message is sent to all neighboring organisms instead of the single faced organism.

Figure 7 plots average and best-case performance of individual demes across all 30 trials. As with the previous experiment, mean deme performance is significantly different from previous results ($p < 0.001$), and again, best-case performance of individual demes is unchanged. Thus, we conclude that the addition of broadcast messaging alone does not aid in the evolution of consensus.

E. Sensing Death

Under the observation that it is the “death” of an individual that signals the start of a new round of consensus, and that these deaths are rare events, we next examined the effect of a background rate of death on the evolution of consensus.

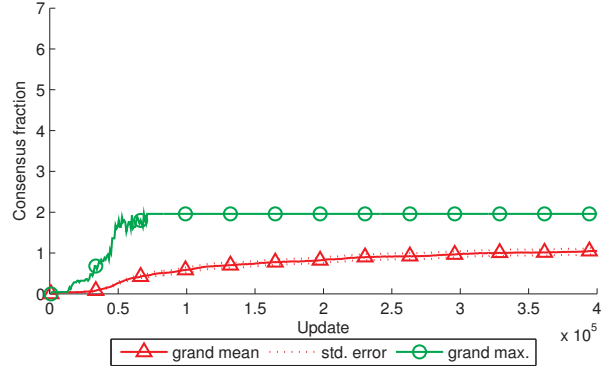


Figure 7. Deme performance on the Iterated Consensus Dilemma, with the addition of broadcast messaging.

Specifically, we established a 0.025% chance per update that a single individual within each deme will be replaced, at which point new cell data is assigned as well. At this rate, on average each deme will experience 20 deaths (out of 25 organisms) during a single competition period.

Figure 8 plots average and best-case performance of individual demes across all 30 trials. In this case, not only is the mean deme performance significantly different than previous results ($p < 0.001$), the best-case performance of an individual deme oscillates between three and four complete consensus rounds, a marked improvement. Based on these results, we conclude that a background rate of death has served to sensitize evolution to the death of individuals, thus making it easier for them to sense the start of a new consensus round.

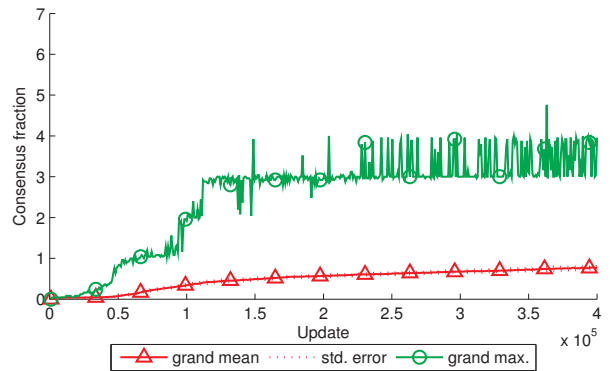
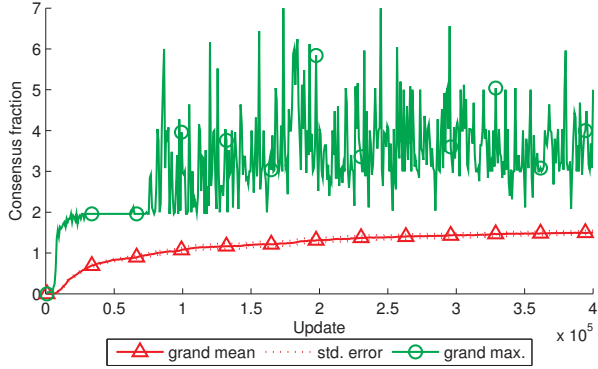


Figure 8. Deme performance and detailed behavior for the Iterated Consensus Dilemma, with the addition of death during deme competition periods.

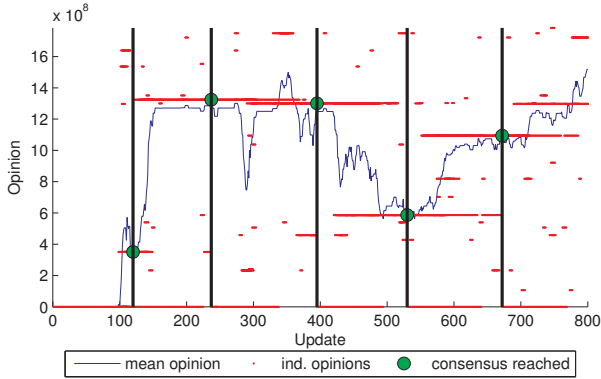
F. Synchronization

In the final experiment presented here, we added synchronization primitives to the instruction set. Specifically, we added the `flash` and `get-flash-info` instructions, which

VI. GENOME ANALYSIS



(a) Average and best-case deme performance.



(b) Representative behavior of a single deme passing multiple consensus rounds.

Figure 9. Deme performance and detailed behavior for the Iterated Consensus Dilemma, with the inclusion of synchronization instructions.

send virtual “flashes” (a la fireflies) to an organism’s neighbors, and retrieve information about any sensed flashes, respectively. We also provide organisms with the `bcast1` instruction, enabling them to broadcast messages to all of their neighbors with a single instruction.

Figure 9(a) plots the average and best-case performance of individual demes across all 30 trials and Figure 9(b) depicts the specific behavior of a single deme, including the individual opinions (points), mean group opinion, when consensus is reached (vertical line), and the value of that consensus (filled circle). Here we see a significant improvement in both the average and best-case deme performance, where the average deme approaches 1.5 consensus rounds, and the best-case performance of any deme achieves five consensus rounds. An interesting result brought to light in this experiment is the role of *historical contingency*, and its relationship to instructions. Specifically, it appears as though a combination of broadcast, sensing, and synchronization instructions were needed as building blocks for the behavior shown in Figure 9(b), although the genome responsible, described below, does not include the `flash` instruction.

In this section we analyze the most-fit dominant genome from the experiment described in Section V-F, the same genome responsible for the behavior in Figure 9(b). Analysis is complicated by the fact that the evolved behavior depends on the random cell data and also on interactions with other individuals within the deme. Hence, testing an organism in isolation provides little insight into its behavior. To overcome these difficulties, three different techniques were used. First, we performed a knockout analysis of each instruction in this genome; second, a detailed analysis of the relevant instructions highlighted by the knockout analysis was conducted; and finally, the putative algorithm was re-implemented and evaluated outside of AVIDA to verify our understanding of the evolved behavior.

A. Knockouts

Due to the cryptic nature of evolved genomes, we first conducted knockout experiments on the genome to identify the instructions most important to consensus. In AVIDA, a knockout mutation is the replacement of an instruction in an organism’s genome with a `nop-X`, an instruction that performs no computation. By examining the effect of a knockout mutation on overall group behavior, we can pinpoint those instructions that contribute to consensus. Specifically, we generated all possible single-point knockouts by replacing each of the instructions in the genome with a `nop-X`. There are 86 instructions in this genome, thus there are 86 possible knockout mutations, and each of these mutants was tested 30 times, resulting in 2,580 total trials. Figure 10 depicts the resulting fitnesses of these trials. For each knockout position, referring to the instruction in the genome that was replaced by a `nop-X`, a box-plot for the 30 resulting fitness values was generated. In this figure, we see that the majority of fitnesses from a knockout fall in the range 0.5×10^4 to 1.0×10^4 , indicating that those knockouts had very little effect on fitness, and thus the replaced instructions were not likely to contribute to consensus. However, knockouts of instructions near position 25, 43, 47, 50-53, and 57 reduced fitness to 0, indicating that these instructions are critical for consensus behavior.

B. Annotated genome

Figure 11 depicts the annotated genome. Instructions that do not affect the overall fitness of this genome are not labeled. We observe that this genome evolved to use a fairly short (36 out of 86 instructions) loop to solve the ICD.

Of particular interest are the instructions at position 26 and 61, which together form a loop around the intervening instructions. Instruction 31 senses cell data, while instructions 33, 35, 44, 48, and 55 retrieve messages that have been sent to the caller. Instruction 52 sets the organism’s opinion, while instruction 57 broadcasts its opinion. The importance

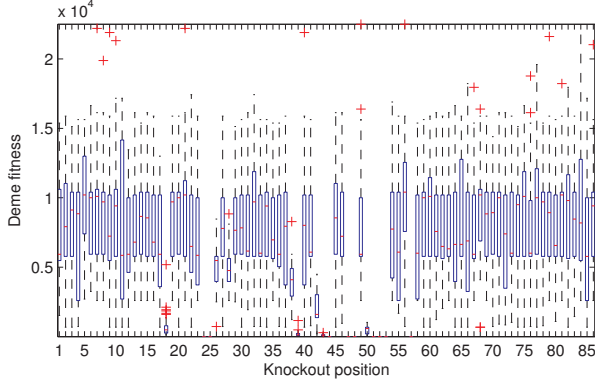


Figure 10. Fitness vs. knockout position, averaged over 30 trials. The most significant instructions are located mid-way through this genome.

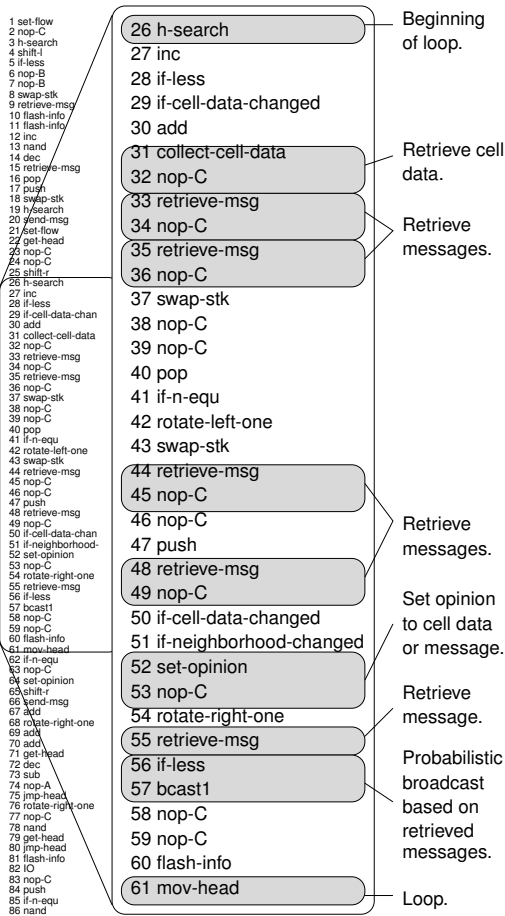


Figure 11. Annotation of the best-performing final dominant from the experiment described in Section V-F.

of these instructions for reaching consensus is supported by the knockout experiments.

C. Evolved Algorithm

Algorithm 1 is pseudocode for the evolved algorithm. The key component of this algorithm is the timing relationship

between the different calls to *retrieveMsg()* and the *if*-statement at line 9. Specifically, these instructions combine to broadcast messages probabilistically and in inverse proportion to the number of messages that are being sent in the organism's neighborhood. Through experimentation both with AVIDA and a simulation of this algorithm written in Python, we found that consensus was achieved when the probability of broadcasting a message was below approximately 25%; consensus was rarely achieved at higher broadcast rates. One implication of this result for distributed algorithms that warrants further study is the seeming contradiction that sending fewer messages leads to a more stable system. Moreover, we observe that this behavior results in decreased synchronicity among organisms, a property that has been shown to increase system stability [27].

Algorithm 1 Evolved algorithm for solving the ICD.

Require: *opinion* is null; $AX, BX, CX = 0$.

loop

- 2: $CX \leftarrow cellData$
 $(CX, AX) \leftarrow retrieveMsg()$
 - 4: $(CX, AX) \leftarrow retrieveMsg()$
 $(CX, AX) \leftarrow retrieveMsg()$
 - 6: $(CX, AX) \leftarrow retrieveMsg()$
 $setOpinion(CX)$
 - 8: $(BX, CX) \leftarrow retrieveMsg()$
if $BX < CX$ **then**
 - 10: $broadcast(CX, AX)$
 - end if**
 - 12: **end loop**
-

VII. DISCUSSION

Based on the analysis of the evolved algorithm from the previous section, we can place it in the class of *randomized asynchronous consensus* protocols [28], where the random arrival times of messages are exploited to achieve consensus. The evolved algorithm itself is based on probabilistic messaging, similar to gossip-based protocols such as that described in [29], and it also bears similarity to a number of algorithms for distributed consensus that use random processes. For example, Aspnes [30] describes *lean-consensus*, a probabilistic algorithm for bit-consensus, where races between processes are exploited to solve consensus; Aysal *et al.* [31] describe a probabilistic time-quantization (PTQ) method for producing a distributed average; and Chandra [21] presents a solution to n -process id consensus based on coin-flipping.

In the course of simulating the lean-consensus and PTQ algorithms for comparison to the evolved strategy, we uncovered a number of important differences. The most significant of these differences is that our model for consensus is *continuous*, and includes the random death of agents. For example, Figure 12 depicts the representative behavior of

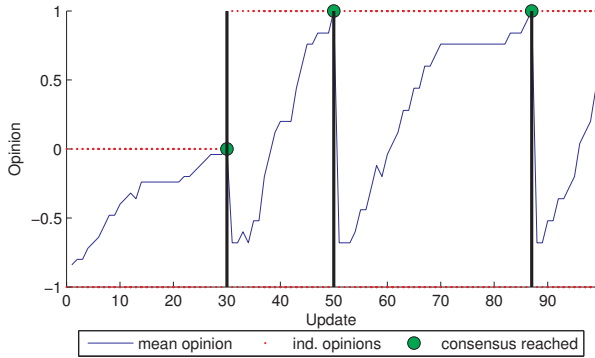


Figure 12. Simulation of lean-consensus [30]. All agents are reset once consensus is reached.

a simulation of the lean-consensus algorithm. Compared to Figure 9(b), this behavior appears “cleaner,” with a steady change in average opinion towards consensus. However, once consensus is reached, this algorithm has no mechanism to recover from the removal of the agreed-upon value. In other words, a system based on lean-consensus has no capacity to “change its mind” once a sufficient number of agents have agreed upon a course of action. Thus, once consensus has been achieved, the state of all agents must be reset in order to subsequently reach consensus. In Figure 12, we manually restart the algorithm once consensus is reached. In contrast, the behavior shown in Figure 9(b) contains a number of oscillations between each consensus, and the death of any agent is allowed. A second difference is found in the communications model. In particular, we assume that agents are able to communicate only with their neighbors (eight, in this study), as opposed to the multi-writer register model used in other studies. This complicates the consensus problem by requiring that organisms evolve their communication protocol as well as the consensus algorithm. As far as we are aware, the evolved algorithm presented here is a novel approach to consensus.

It is possible that some of the techniques designed for the analysis of traditional probabilistic algorithms could be used to better understand evolved algorithms. For example, techniques developed for understanding randomized distributed algorithms in general [32], as well as asynchronous randomized consensus in particular [33], could improve our understanding of the evolved algorithm presented here. Moreover, Olfati-Saber *et al.* [22] describe a general framework by which consensus algorithms can be studied, though some adaptation for randomized processes would likely be required.

Broadly speaking, using evolutionary computation to develop distributed algorithms enables the developer to take into account many of the challenges facing designers of modern distributed systems. For example, as part of this study, we examined the effects of granting organisms the

capability of sensing node turnover, as well as instructions for broadcasting messages and synchronization. We additionally verified that the evolved behaviors scaled to larger population sizes, and conducted experiments that focused on the stability of consensus, where we required that all organisms within a deme maintain a common opinion for consecutive updates. The details of these additional experiments can be found in [34]. Finally, the addition of message loss or corruption into the AVIDA system could supply us with an algorithm for consensus that is resilient to these hazards, just as the inclusion of MANET-specific concerns, for example, battery conservation, into fitness functions would enable the simultaneous evolution of distributed behavior and optimization of resource utilization.

VIII. CONCLUSION

The experiments described in this paper demonstrate that digital evolution can be used to evolve novel distributed behaviors for reaching consensus. Based on the results of different experimental treatments, we can conclude that the availability of synchronization and broadcast primitives, as well as a low rate of population turnover, have a significant impact on the evolvability of distributed behavior, even if those features are not used in the final solutions. Moreover, we have shown that digital evolution is capable of using features of the environment, in this case, message arrival times, to produce cooperative behaviors. The results presented in this paper also demonstrate a near-complete development life-cycle for using digital evolution as a tool in the design of distributed algorithms, including initial evolution, analysis, and simulation stages.

Ongoing and future work include examining the stability of consensus, where instead of holding a common value for at least one update, we require organisms to share the same value for a longer period of time; consensus in irregular or self-constructed network topologies; and the introduction of various network and environmental hazards in order to discover strategies to deal with adverse conditions.

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] C. List, “Democracy in animal groups: a political science perspective,” *Trends in Ecology and Evolution*, vol. 19, no. 4, pp. 168–169, 2004.
- [2] D. J. T. Sumpter, J. Krause, R. James, I. D. Couzin, and A. J. W. Ward, “Consensus decision making by fish,” *Current Biology*, vol. 18, no. 22, pp. 1773–1777, 2008.

- [3] C. M. Waters and B. L. Bassler, "Quorum sensing: Cell-to-cell communication in bacteria," *Annual Review of Cell and Developmental Biology*, vol. 21, no. 1, pp. 319–346, 2005.
- [4] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Computing Surveys (CSUR)*, vol. 25, no. 2, pp. 171–220, 1993.
- [5] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2006.
- [6] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.
- [7] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1997.
- [8] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "NASA's swarm missions: The challenge of building autonomous software," *IT Professional*, vol. 06, no. 5, pp. 47–52, 2004.
- [9] C. Ofria and C. Adami, "Evolution of genetic organization in digital organisms," in *Proceedings of DIMACS Workshop on Evolution as Computation*, 1999.
- [10] 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.
- [11] D. B. Knoester, P. K. McKinley, and C. Ofria, "Cooperative network construction using digital germlines," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2008.
- [12] B. E. Beckmann and P. K. McKinley, "Evolution of adaptive population control in multi-agent systems," in *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2008, pp. 181–190.
- [13] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.
- [14] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.
- [15] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2007, pp. 398–407.
- [16] T. Herman, "Models of self-stabilization and sensor networks," in *Proceedings of the International Workshop on Distributed Computing (IWDC)*, 2003, pp. 205–214.
- [17] C. F. Camerer, *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press, 2003.
- [18] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*. Springer, 2007.
- [19] G. Baldassarre, D. Parisi, and S. Nolfi, "Distributed coordination of simulated robots based on self-organization," *Artificial Life*, vol. 12, no. 3, pp. 289–311, 2006.
- [20] K. Wagner, J. A. Reggia, J. Uriagereka, and G. S. Wilkinson, "Progress in the simulation of emergent communication and language," *Adaptive Behavior*, vol. 11, no. 1, pp. 37–69, 2003.
- [21] T. D. Chandra, "Polylog randomized wait-free consensus," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 1996, pp. 166–175.
- [22] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, 2007.
- [23] R. M. Axelrod and W. D. Hamilton, "The evolution of cooperation," *Science*, vol. 211, no. 4489, pp. 1390–1396, 1981.
- [24] K. Wagner and J. A. Reggia, "Evolving consensus among a population of communicators," *Complexity International*, vol. 9, 2002.
- [25] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu.com, 2008.
- [26] D. B. Knoester, P. K. McKinley, B. E. Beckmann, and C. Ofria, "Directed evolution of communication and cooperation in digital organisms," in *Proceedings of the European Conference on Artificial Life (ECAL)*, 2007.
- [27] A. Campbell and A. S. Wu, "On the significance of synchronicity in emergent systems," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
- [28] J. Aspnes, "Randomized protocols for asynchronous consensus," *Distributed Computing*, vol. 16, no. 2-3, pp. 165–175, 2003.
- [29] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, 2005.
- [30] J. Aspnes, "Fast deterministic consensus in a noisy environment," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2000, pp. 299–308.
- [31] T. C. Aysal, M. Coates, and M. Rabbat, "Distributed average consensus using probabilistic quantization," in *Proceedings of the IEEE/SP Workshop on Statistical Signal Processing (SSP)*, 2007, pp. 640–644.
- [32] N. A. Lynch, I. Saia, and R. Segala, "Proving time bounds for randomized distributed algorithms," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 1994, pp. 314–323.
- [33] H. Attiya and K. Censor, "Tight bounds for asynchronous randomized consensus," *Journal of the ACM (JACM)*, vol. 55, no. 5, pp. 1–26, 2008.
- [34] D. B. Knoester and P. K. McKinley, "Evolution of probabilistic consensus in digital organisms," Computer Science and Engineering, Michigan State University, East Lansing, Michigan, Tech. Rep. MSU-CSE-09-13, April 2009, <http://www.cse.msu.edu/thinktank/consensus-TR.pdf>.