Evolution of Probabilistic Consensus in Digital Organisms Technical Report: MSU-CSE-09-13

David B. Knoester and Philip K. McKinley Department of Computer Science and Engineering Michigan State University East Lansing, Michigan 48824 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 biologicallyinspired approaches to designing robust distributed systems. Examples include biomimetics, which mimic behaviors such as swarming found in nature, as well as evolutionary computation methods, such as genetic algorithms and neuroevolution, which simulate the natural processes that produce those behaviors. A related but fundamentally different technique is digital evolution, a type of artificial life system whereby a population of self-replicating computer programs exists in a user-defined computational environment and is subject to instruction-level mutations and natural selection. Over thousands of generations, these digital organisms can evolve to survive, and thrive, under extremely dynamic and adverse conditions. In this paper, we describe a study in the use of digital evolution 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.

1. Introduction

There are many examples of organisms in nature that exhibit cooperative behaviors of varying complexity. For example, nearly all species of microorganism cooperate to form extracellular structures called biofilms [22], and many social insects cooperate to build complex nests [27]. Some cooperative behaviors exhibit *consensus*, where members of a group agree upon a particular course of action. Consensus behaviors are visible throughout the spectrum of life, from decisionmaking in humans [37], leader election in schools of stickleback fish [50], alarm-drumming in ants [29], and quorum sensing in bacteria [56]. Generally, consensus among individuals in a group leads to improved decisions [21]. 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 [10] or provide the basis for distributed lock management [15]. However, due to the fact that bounded-time consensus in the presence of failures has been proven impossible (the so-called FLP impossibility proof) [25, 38], numerous heuristics for consensus have been developed [40]. As computing systems continue to expand their reach into the natural world, for example through cyber-physical systems [58], as well as scaling up in size and complexity, designers are faced with a multitude of additional complications (such as the shift from systems comprised of few highperformance nodes to large systems of unreliable, commodity hardware [41]), 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. As such, many recent approaches to designing distributed systems have focused on *biomimetics*, where behaviors observed in nature are replicated in silico [51]. 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, without a priori knowledge of 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 [34].

In this study, we use *digital evolution* [42], a form of evolutionary computation, to evolve digital organisms that exhibit consensus. In digital evolution, a population of digital organisms 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 [44], a digital evolution platform previously used to study biological evolution [36]. 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 generation of software behavioral models that assist the developer in addressing uncertainty in dynamically adaptive systems [26], to the discovery of algorithms for leader election and construction of communication networks [31, 32].

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 were able to understand its behavior, which was based on probabilistic message forwarding. Third, we implemented this algorithm in a simulator 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.

We note that this approach is *complementary* to other advances in distributed systems. For example, existing algorithms for routing and constructing overlay networks can be provided as building blocks within an evolutionary computation framework for designing distributed applications, further increasing our ability to design for complex environments. Our research explores the role digital evolution can play in designing distributed systems that remain effective under dynamic conditions. While evolved solutions may share some of the inherent imperfections of natural organisms, they might also be resilient to unexpected conditions, in much the same way that living organisms adapt to their environment.

2. Related Work

Replication of the critical components of a distributed computing system is a well-known method to improve overall system reliability [53, 45, 49]. Consensus algorithms are frequently used to ensure that state information shared among these replica remains consistent [10, 40]. In practice, algorithms such as Paxos [35] are used as the basis for implementations of consensus [19], which in turn can support higher-level services such as distributed lock management [15]. 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 environment in which these systems are deployed. Indeed, numerous distributed algorithms already integrate techniques from dynamic systems [28, 24] to mitigate the effects of dynamic environments.

The consensus problem specifically has been approached from many different fields of research. While the motivation for this paper stems from consensus in distributed systems, consensus itself is studied in fields as diverse as coordination games [17], multi-agent systems [47], and cooperative control [48]. Coordinating the behavior of multiple agents is also a common problem in evolutionary robotics, as in [8, 52], and many studies in artificial life have contributed to our understanding of the evolution of cooperation and communication [55, 9].

In this study, we use a model of the consensus problem based on [40] and [18], called the *n*-process id consensus problem, in which a group of processes seek to reach agreement, or consensus, upon a common process id. 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 7. Given the fundamental usefulness of consensus in distributed algorithms, and its theoretical impossibility [25], 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 [4], or the evolution of communication under fixed cooperation properties [16, 54]. In this study, 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.

3. Digital Evolution and Avida

Digital evolution [43] is a form of evolutionary computation that was originally developed to study evolution in biology. AVIDA [44], a platform for digital evolution, has also recently been applied to engineering problems, including the design of dynamically adaptive systems [26], construction of communication networks [32], information gathering and energy management [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 [46]. 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 for application to computing systems. First, digital organisms in AVIDA have only rudimentary computation capabilities, comparable to resourceconstrained 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 [33]. Third, AVIDA includes features that enable the evolution of group behaviors. In the remainder of this section, we briefly describe the structure of an organism in AVIDA and the mechanism by which group behaviors can be evolved.

Recently, we have shown that digital evolution is capable of evolving distributed problem solvers. For example, in [31] we used digital evolution to evolve a novel algorithm for leader election, where the leader was elected based on the characteristics of its genome. In that study, the strategies employed depended on characteristics of the AVIDA environment, specifically self-replication, mutation, and direct inspection of genomes. In this study, we remove the ability for organisms to rely on these features, and evolve behaviors that only depend on message sending.

3.1. 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 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*). Different AVIDA CPU architectures have been implemented and used in various studies [44]. 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 and stack pointer.



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

Instructions within an organism's genome are similar in appearance to a traditional assembly language. These instructions enable an organism to perform sim-

ple mathematical operations, such as addition and multiplication; to manipulate heads within their genome; to sense and change properties of the environment; and to communicate with neighboring organisms. Certain instructions also enable organisms to replicate, subject to mutation, and thus spread throughout the population. Instructions in AVIDA can also have different costs (in terms of virtual CPU cycles) associated with them. For example, a simple addition may cost only one cycle, while broadcasting a message may cost 20 cycles. A key property of AVIDA's instruction set that differs from traditional computer languages is that it is not possible to construct a syntactically incorrect genome in AVIDA. Hence, while random mutations may produce many genomes that do not perform any meaningful computation, their instruction sequences are always executable.

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 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.

3.2. Levels of Selection

Natural selection is the differential survival and reproduction of organisms within their environment [20], while multilevel selection is the theory that the survival of the individual is linked to the survival of its group [57]. These groups may be defined in many different ways, for example by a common trait (a traitgroup), shared ancestry (clade selection), membership in the same species (species selection), or by the interactions between related individuals (kin 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. Competition between demes can either be *synchronous*, where periodic selection is applied based on a deme-level fitness function using fitness-proportional selection, or *asynchronous*, where the behavior of each deme is monitored for a userdefined event that triggers the deme's replication. Normally, when a deme replicates, all of its constituent organisms are replicated as well. 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* [32]. As in the group selection case, replication when using a germline can be either synchronous or asynchronous.



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

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 periodically calculates the fitness of each deme via a user-defined fitness function. This fitness function takes as input a single deme and produces the fitness of that deme (a floating-point number) as output. Using the resulting array of fitness values, the CompeteDemes framework then performs fitness-proportional selection, preferentially replicating those demes with higher fitness. For 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 better approaches to reaching consensus. At the end of an AVIDA trial, the dominant, or most prolific, genome can be identified for further study, for example to better understand its behavior.

4. 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, for example, 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.

4.1. Instructions

All relevant instructions employed in this study are summarized in Table 1. Of particular note are the instructions associated with messaging, opinions, and the "flash" capability. These instructions enable organisms to send and retrieve messages; manipulate the "opinion" register, used as the basis for the fitness functions that will be presented in Section 5; and to both sense and trigger virtual "flashes," a synchronization primitive based on the behavior of fireflies, respectively.

4.2. Configuration

We configured AVIDA to use the **CompeteDemes** process, outlined in Section 3.2, with specific configuration values summarized in Table 2. We used 400 demes, each comprising 25 digital organisms connected in a torus topology, with each deme configured to use a germline to provide homogeneity within demes. The relatively small size of each deme was selected primarily for practical (i.e., computation time) purposes, and the default values were used for all mutation rates.

In order to control the initial conditions for each deme, we configured AVIDA to disallow self-replication of individual organisms. Each time a deme replicated as a result of the CompeteDeme process, the offspring deme was filled with 25 copies of the latest (possibly

Table 2. Common Avida configuration	ons used
for the experiments described in this	s 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 us-
	ing fitness-proportional
	selection (periodic, every
	800 updates).

mutated) genome from the germline, the genome attached to the deme. As the final step of AVIDA configuration, we defined the *default ancestor*, which is the starting organism for each deme within all AVIDA trials. Here, we used an organism that contained 100 nop-C instructions; the nop-* instructions perform no computation, and do not change the state of the virtual CPU. The presence of a large number of nop-C instructions in the default ancestor is common in AVIDA experiments, and provides evolution with a "blank tape" for mutating different instructions into the genome. We emphasize that although the default ancestor contains only 100 instructions in its genome, not only can mutations increase and decrease genome size, the genome itself is circular; once the organism executes the final instruction, execution flow wraps around to the beginning of the genome.

5. Results

5.1. Simple Consensus

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

Each agent within a group is assigned a unique identifier. Agents within this group are independent, and can communicate by

¹This presentation of the consensus problem is intentionally patterned after the classic Prisoner's Dilemma. Game theoretic studies have historically provided insight into many of the fundamental aspects of evolution, such as cooperation [5].

Instruction	Description
send-msg	Sends a message to the neighbor currently faced by the caller; message contains
	contents of BX and CX registers.
retrieve-msg	Loads the caller's BX and CX registers from a previously received message.
rotate-left-one	Rotate this organism counter-clockwise one step.
rotate-right-one	Rotate this organism clockwise one step.
get-opinion	Sets register PX ? to the value of the caller's opinion register.
set-opinion	Sets the caller's opinion register to the value in register BX .
bcast1	Sends a two-word message containing the values of registers BX and CX to all
	immediately neighboring organisms.
collect-cell-data	Sets register BX to the value of the cell data where the caller lives.
if-cell-data-changed	Execute the subsequent instruction if the cell data where the caller lives has changed
	since the previous call to collect-cell-data, otherwise skip the subsequent instruction.
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 get-neighborhood 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, execute the subsequent
	instruction. Otherwise, skip the subsequent instruction.
flash-info	If the caller has ever received a flash, set BX to 1 and CX to the number of cycles
	since that flash was received. Otherwise, set BX and CX to 0.

Table 1. Relevant instructions for this study. All instructions are equally likely to be selected as targets for mutation.

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 within the group express different opinions, the group perishes. How should the agents act?

The question we are asking here then, is can evolution evolution solve the SCD, and if so, what solutions will be discovered? Using the Opinion, CellData, and CompeteDemes frameworks described earlier, we configured each deme as a 5×5 torus of cells. Each of these cells were assigned a random 32-bit integer as cell data, and we used a fitness function to reward demes whose constituent organisms set their opinion to a common value. The specific fitness function used here was:

$$F = (1 + S_{max})^2$$
 (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, where that opinion also corresponds to a cell value present within the deme. We emphasize that although organisms are

able to set their opinion to any integer value, only opinions that are set to a cell data present within the deme contribute to the deme's fitness.

Figure 3(a) plots 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 xaxis is in units of *updates*, the standard unit of time in AVIDA. Here we see that the best-performing demes achieve consensus after approximately 25,000 updates, while the average deme steadily approaches consensus. Figure 3(b) depicts a representative behavior of a single deme from this experiment. Here, individual opinions are shown as points in red, while the mean opinion over all individuals within the deme is shown in blue. Consensus among all individuals is represented as both a vertical black line and green circle. As shown here, when consensus is reached individual opinions and the mean opinion converge to the same value.

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 [30]. In that previous study, we used



(a) Average and best-case deme performance, in terms of consensus. Consensus is first achieved near update 25,000.



(b) Representative behavior of a single deme while reaching consensus. This behavior was produced by testing the dominant (most prolific) genome from one of 30 trials.

Figure 3. Deme performance and detailed behavior for the Simple Consensus Dilemma.

AVIDA *tasks*, a mechanism to apply selective pressure 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 on the group of individuals, 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 data contained in its own cell, or the contents of a received message, whichever is larger. Similarly, when a group of organisms all share this genome, each will eventually set its opinion to the maximum cell data that any of the group members have access to.



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

5.2. Iterated Consensus

In the next series of experiments we explored the iterated consensus dilemma. The Iterated Consensus Dilemma (ICD) modifies 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, and then recover to reach consensus again. ICD may be thought of as non-stationary optimization [14], 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 from Section 5.1. The specific fitness function we used was:

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

where F is the resulting fitness, S_{max} is again 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). In order 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, and the cell data corresponding to the agreed-upon value is reset to a random 32-bit integer. Whenever a cell data is reset, the new value is selected from the range bounded by the maximum and minimum cell data present within that deme.

Figure 5(a) plots average and best-case performance of individual demes across all 30 trials. Here we see that the average deme approaches a single consensus round, while the best-case performance of any deme is two consensus rounds. Figure 5(b) depicts the behavior of a deme that achieved two consensus rounds. The remaining experiments examine various ways in which evolution might improve solutions to the ICD.



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 50,000.



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



5.3. Neighborhood Sensing

In this experiment, we provide additional instructions that organisms may use to sense their neighborhood for changes. Specifically, we add the getneighborhood 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, respectively. Figure 6(a) plots average and best-case performance of individual demes across all 30 trials, and Figure 6(b) depicts the detailed behavior of a single deme from this experiment. Although the mean behavior of demes from this experiment is significantly different from that in Section 5.2 (p < 0.001, Mann-Whitney U-test), and the specific behavior of individual demes is qualitatively different, 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.



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 60,000.



(b) Representative behavior of a single deme passing multiple consensus rounds with the addition of neighborhood-sensing.

Figure 6. Deme performance and detailed behavior for the Iterated Consensus Dilemma, with the addition of neighborhood-sensing instructions.

5.4. 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(a) plots average and best-case performance of individual demes across all 30 trials. As with the previous experiment, the individual deme behavior shown in Figure 7(b) is qualitatively different from previous results, and mean deme performance is significantly different from previous results (p < 0.001), however best-case performance of individual demes is no different than previous results. Thus, we conclude that the addition of broadcast messaging alone does not aid in the evolution of consensus.

5.5. 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 small background rate of death on the evolution of consensus. Specifically, here we establish a 0.025% chance per update of an individual within a deme to be replaced, including new cell data. At this rate, on average each deme will experience 20 deaths during a single competition period.

Figure 8(a) plots average and best-case performance of individual demes across all 30 trials, and Figure 8(b) depicts the specific behavior of a single deme. 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 significant improvement. Based on these results, we conclude that a small background rate of death has served to sensitize demes to the death of individuals, thus making it easier for them to recover from death.

5.6. Synchronization

In the next experiment, we add synchronization primitives to the instruction set. Specifically, we add the flash and get-flash-info instructions, which send virtual "flashes" (a la fireflies) to an individual's neighbors, and retrieve information about any sensed flashes, respectively. Here we also provide individuals with the bcast1 instruction, enabling them to send messages that match the distribution of the virtual flashes.



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 60,000.



(b) Representative behavior of a single deme passing multiple consensus rounds with the addition of broadcasting.

Figure 7. Deme performance and detailed behavior for the Iterated Consensus Dilemma, with the addition of an instruction enable broadcast messages.

Figure 9(a) plots average and best-case performance of individual demes across all 30 trials, and Figure 9(b) depicts the specific behavior of a single deme. 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 bestcase deme at the end of all trials 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 are needed as building blocks for the behavior shown in Figure 9(b), although the genome responsible does not include the flash instruction.



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 100,000.



(b) Representative behavior of a single deme passing multiple consensus rounds, with the addition of death.

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

5.7. Stability of Consensus

In our final experiment, we demonstrate the flexibility of an evolutionary computation approach to designing distributed behaviors. In previous experiments, we required that all organisms express the same opinion once, and during the same update. In this experiment we alter the fitness function to reward for *stability* of consensus. Specifically, we require all organisms in the same deme to express the same opinion for a series of consecutive updates – In other words, we reward for *stable consensus*. The specific fitness function we used was:

$$F = (1 + S_{max} + D(R \cdot H + L_{max}))^2$$
(3)

where F is the resulting fitness; S_{max} is again the maximum support; R is the number of times the deme has reached consensus during this competition period; H



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 20,000.



(b) Representative behavior of a single deme passing multiple consensus rounds, with the inclusion of synchronization instructions.

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

is the *hold time*, a configurable value for the number of updates that all organisms in the deme must share the same opinion before we consider consensus to have been reached; and D is the size of the deme (again, 25). We found the final component of this fitness function, L_{max} , to be critical for the evolution of stable consensus. Here, L_{max} is the maximum level, and is defined as the maximum number of subsequent updates during which organisms in a deme shared the same opinion without reaching consensus. For example, if we set the hold time (H) to four, and during a competition period all organisms in a deme express the same opinion during subsequent updates for at most three updates, then L_{max} is three. Were this deme to hold that same opinion for one more update, we then count consensus as having been reached, increment R, and reset L_{max}

to zero for the next round.

In this experiment, we test three different values of H: 2, 4, 8, 16, and 32 updates, and perform 30 trials of each. Figure 10(a) plots average and best-case performance of individual demes across all 30 trials for a hold time of 32 updates (H = 32), and Figure 10(b) depicts the specific behavior of a single deme. Compared to the results in Figure 9, fewer consensuses are reached, which is to be expected, as this is a more difficult problem where each consensus requires more time.



(a) Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 20,000.



(b) Representative behavior of a single deme passing multiple consensus rounds, with a consensus hold time of 32 updates.

Figure 10. Deme performance and detailed behavior for Stable Consensus.

6. Genome Analysis

In this section we analyze the dominant genome from the experiment described in Section 5.6, the same genome responsible for the behavior in Figure 9(b). Analysis is complicated by the fact that the evolved behavior depends not only the random cell data, but 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 reimplemented and evaluated outside of AVIDA to verify our understanding of the evolved behavior.

First, however, we describe how we selected this genome for closer analysis. We first extracted the final dominant genome from the original 30 trials of the experiment described in Section 5.6. We then conducted 30 isolated trials of each of these 30 dominant genomes, where each trial consisted of executing a single deme, and recorded the resulting fitness of that deme based on the fitness function in Equation 2. We selected the genome corresponding to the highest average fitness for analysis.

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, and tested each of the resulting mutants 30 times. There are 86 instructions in this genome, thus there are 86 possible knockout mutations, and each of these mutants were testing 30 times, resulting in 2,580 total trials. Figure 11 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.

Annotated genome Figure 12 depicts the annotated genome. In this figure, each instruction has been color-coded for whether it interacts with its neighbors or environment, whether it depends directly on the ac-



Figure 11. Fitness vs. knockout position, averaged over 30 trials. The most significant instructions are located mid-way through this genome.

tivities of its neighbors, or whether it is an individual behavior. Instructions that do not affect the overall fitness of this genome have been marked as neutral. Interestingly, this genome has evolved to utilize a fairly short (36 out of 86 instructions) loop to solve the ICD.

Of particular interest in this genome 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 to its neighborhood. These observations are supported by the knockout experiments.

Evolved Algorithm To test that the evolved algorithm was general, we configured AVIDA with larger demes (100 organisms, four times larger than that used to evolve this genome). Figure 13 depicts the behavior of a representative trial with this larger population size. As can be seen here, consensus is still reached, however it takes significantly longer for the population to converge (725 vs. 120 updates).

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 nine. 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, we found that consensus was achieved when the probability of broadcasting a message was below approximately 25%,



Figure 12. Annotation of the best-performing final dominant from the experiment described in Section 5.6.

while 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.

Figure 14 shows the results of a simulation of this algorithm written in Python. As can be seen here, consensus was reached twice in 100 updates. We note that for this simulation, an *update* refers to a single iteration of the loop in Algorithm 1 for all processes. This



Figure 13. Representative behavior of the genome from Figure 12 when tested in a deme four times larger (100 organisms) than that it was evolved in, showing generality of the evolved genome.

Alg	gorithm 1 Evolved algorithm for solving the ICD.
Re	quire: 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

is a different definition of update than in AVIDA, where each individual executed only five virtual CPU instructions (in other words, an update in Figure 14 is *longer* than an update in AVIDA). Given that the length of the loop in Figure 12 is 35 instructions, this corresponds to a 1:7 ratio between updates (that is, one simulation update corresponds to seven AVIDA updates).

7. 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 [2, 13], where the random arrival times of messages are exploited in order to achieve consensus. The evolved algorithm bears similarity to at least three algorithms for distributed consensus: In [6], Aysal, Coates, and Rab-



Figure 14. Detailed behavior of a simulation of Algorithm 1.

bat describe a probabilistic time-quantization method for producing a distributed average. In [1], Aspnes describes *lean-consensus*, a probabilistic algorithm for bit-consensus, where races between processes are exploited to solve consensus. Finally, in [18], Chandra presents a solution to *n*-process id consensus based on coin-flipping.

In the course of simulating these 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 *contin*uous, and includes random death of agents. For example. Figure 15 depicts the representative behavior of a simulation of Aspnes' lean-consensus algorithm. At first glance, 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 contrast, the behavior shown in Figure 10(b) contains a number of oscillations between each consensus, and death of agents, even the agent owning the current maximallysupported opinion, is allowed. A second difference is found in the communications model. In particular, we assume that agents are able to communicate only with neighboring individuals (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 know, the evolved algorithm presented here is a novel approach to consensus.



Figure 15. Simulation of lean-consensus [1]. All agents are reset once consensus is reached.

In a more general sense, using evolutionary computation to develop distributed algorithms enables us to take into account many of the challenges facing modern distributed systems by incorporating these challenges directly into the environment in which evolution takes place. For example, 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. Moreover, the inclusion of MANET-specific concerns, for example, battery conservation, into fitness functions would enable the simultaneous optimization of multiple criteria.

It is also possible that techniques designed for the analysis of traditional probabilistic distributed algorithms could be used to better understand evolved algorithms. For example, in [3], Attiya and Censor established tight time bounds for asynchronous randomized consensus, which may hold for evolved algorithms. It may also be the case that techniques developed for understanding randomized distributed algorithms could be used here [39].

Finally, we note a behavior exhibited by many of the evolved algorithms that suggests evidence of self-organization. In many of the figures containing detailed behaviors, especially Figure 8(b) and Figure 10(b), we see a common response to the removal of the source of the agreed-upon value immediately following consensus. Specifically, the average opinion changes abruptly. This behavior suggests that equilibrium (consensus) exists on the threshold between stability and instability – a claim that is supported by the instability immediately following consensus. Further study is required, but it is possible that evolutionary algorithms are capable of producing behaviors that incorporate self-organized criticality [7], where the desired behavior of the system can be encoded such that it is resilient to changes in parameters, similar to selfstabilizing algorithms [23].

8. Conclusion

The experiments described here show that digital evolution can be used to evolve 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 a low rate of population turnover, have a significant impact on the evolvability of effective distributed behavior, even if those features are not used in the final solutions. Moreover, we have shown that digital evolution is capable of using novel features of the environment, in this case, message arrival times, to produce probabilistic 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 includes 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; introducing various network and environmental hazards in order to discover strategies to deal with adverse conditions; and competitive coevolutionary studies, where groups of individuals are forced to compete with adversaries for resources.

References

- J. Aspnes. Fast deterministic consensus in a noisy environment. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pages 299–308, 2000.
- [2] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.
- [3] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM* (*JACM*), 55(5):1–26, 2008.
- [4] R. M. Axelrod. The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration. Princeton University Press, 1997.
- [5] R. M. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.
- [6] 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), pages 640–644, 2007.

- [7] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality. *Phys. Rev. A*, 38(1):364–374, Jul 1988.
- [8] G. Baldassarre, D. Parisi, and S. Nolfi. Distributed coordination of simulated robots based on self-organization. *Artificial Life*, 12(3):289–311, 2006.
- [9] C. Baray. Evolving cooperation via communication in homogeneous multi-agent systems. In Proceedings of the International Conference on Intelligent Information Systems, 1997.
- [10] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault-tolerant computing. ACM Computing Surveys (CSUR), 25(2):171–220, 1993.
- [11] B. Beckmann, P. K. McKinley, D. B. Knoester, and C. Ofria. Evolution of cooperative information gathering in self-replicating digital organisms. In *Proceed*ings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO), July 2007.
- [12] B. E. Beckmann and P. K. McKinley. Evolution of adaptive population control in multi-agent systems. In Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pages 181–190, 2008.
- [13] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.
- [14] J. Branke. Evolutionary Optimization in Dynamic Environments. Springer, 2001.
- [15] M. Burrows. The chubby lock service for looselycoupled distributed systems. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, 2006.
- [16] P. C. Buzing, A. Eiben, and M. C. Schut. Emerging communication and cooperation in evolving agent societies. *Journal of Artificial Societies and Social Simulation*, 8(1), 2005.
- [17] C. F. Camerer. Behavioral game theory: Experiments in strategic interaction. Princeton University Press, 2003.
- [18] T. D. Chandra. Polylog randomized wait-free consensus. In Proceedings of the ACM symposium on Principles of Distributed Computing (PODC), pages 166– 175, 1996.
- [19] 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), pages 398–407, New York, NY, USA, 2007. ACM.
- [20] J. K. Conner and D. L. Hartl. A Primer of Ecological Genetics. Sinauer, 2004.
- [21] L. Conradt and T. J. Roper. Consensus decision making in animals. Trends in Ecology and Evolution, 20(8):449–456, 2005.
- [22] M. E. Davey and G. A. O'Toole. Microbial biofilms: from ecology to molecular genetics. *Microbiology and Molecular Biology Reviews*, 64(4):847–867, 2000.
- [23] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. Communications of the ACM, 17(11):643-644, 1974.

- [24] S. Dolev and T. Hermane. Superstabilizing protocols for dynamic distributed systems. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), page 255, 1995.
- [25] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
- [26] H. J. Goldsby, B. H. C. Cheng, P. K. McKinley, D. B. Knoester, and C. Ofria. Digital evolution of behavioral models for autonomic systems. In *Proceedings of* the International Conference on Autonomic Computing (ICAC), 2008.
- [27] D. Grimaldi and M. S. Engel. Evolution of the Insects. Cambridge University Press, 2005.
- [28] T. Herman. Models of self-stabilization and sensor networks. In Proceedings of the International Workshop on Distributed Computing (IWDC), pages 205– 214, 2003.
- [29] B. Holldobler and E. O. Wilson. *The Ants.* Harvard University Press, 1990.
- [30] D. B. Knoester, P. K. McKinley, B. Beckmann, and C. Ofria. Directed evolution of communication and cooperation in digital organisms. In *Proceedings of the European Conference on Artificial Life (ECAL)*, 2007.
- [31] D. B. Knoester, P. K. McKinley, and C. Ofria. Using group selection to evolve leadership in populations of self-replicating digital organisms. In *Proceedings* of the Genetic and Evolutionary Computation Conference (GECCO), 2007.
- [32] 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.
- [33] D. B. Knoester, A. J. Ramirez, P. K. McKinley, and B. H. C. Cheng. Evolution of robust data distribution among digital organisms. In *Proceedings of the Genetic* and Evolutionary Computation Conference (GECCO), 2009. (To appear).
- [34] 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.
- [35] L. Lamport. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133–169, 1998.
- [36] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami. The evolutionary origin of complex features. *Nature*, 423:139–144, 2003.
- [37] C. List. Democracy in animal groups: a political science perspective. Trends in Ecology and Evolution, 19(4):168–169, 2004.
- [38] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. Advances in Computing Research, 4:163– 183, 1987.
- [39] N. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings*

of the ACM Symposium on Principles of Distributed Computing (PODC), pages 314–323, 1994.

- [40] N. A. Lynch. Distributed Algorithms. Morgan Kaufmann, 1997.
- [41] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 840, 2004.
- [42] C. Ofria and C. Adami. Evolution of genetic organization in digital organisms. In *Proceedings of DIMACS* Workshop on Evolution as Computation, 1999.
- [43] C. Ofria, C. Adami, T. C. Collier, and G. K. Hsu. The evolution of differentiated expression patterns in digital organisms. *Lect. Notes Artif. Intell.*, 1674:129– 138, 1999.
- [44] C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Journal of Artificial Life*, 10:191–229, 2004.
- [45] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the* ACM (JACM), 27(2):228–234, 1980.
- [46] R. Poli, W. B. Langdon, and N. F. McPhee. A Field Guide to Genetic Programming. Lulu.com, 2008.
- [47] W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, Jan 2005.
- [48] W. Ren and R. W. Beard. Distributed Consensus in Multi-vehicle Cooperative Control. Springer, 2007.
- [49] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys (CSUR), 22(4):299–319, 1990.
- [50] D. J. T. Sumpter, J. Krause, R. James, I. D. Couzin, and A. J. W. Ward. Consensus decision making by fish. *Current Biology*, 18(22):1773–1777, 2008.
- [51] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. NASA's swarm missions: The challenge of building autonomous software. *IT Professional*, 06(5):47–52, 2004.
- [52] E. Tuci, C. Ampatzis, F. Vicentini, and M. Dorigo. Evolving homogeneous neurocontrollers for a group of heterogeneous robots: Coordinated motion, cooperation, and acoustic communication. *Artificial Life*, 14(2):157–178, 2008.
- [53] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.
- [54] K. Wagner and J. Reggia. Evolving consensus among a population of communicators. *Complexity International*, 9, 2002.
- [55] K. Wagner, J. A. Reggia, J. Uriagereka, and G. S. Wilkinson. Progress in the simulation of emergent communication and language. *Adaptive Behavior*, 11(1):37–69, 2003.
- [56] C. M. Waters and B. L. Bassler. Quorum sensing: Cell-to-cell communication in bacteria. Annual Review of Cell and Developmental Biology, 21(1):319– 346, 2005.

- [57] D. S. Wilson. Introduction: Multilevel selection theory comes of age. *The American Naturalist*, 150:S1–S4, July 1997.
- [58] W. Wolf. Cyber-physical systems. *IEEE Computer*, 42(3), 2009.