EVOLUTION OF DISTRIBUTED BEHAVIOR

By

David B. Knoester

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science
Ecology, Evolutionary Biology, & Behavior

2011

ABSTRACT

EVOLUTION OF DISTRIBUTED BEHAVIOR

By

David B. Knoester

In this dissertation, we describe a study in the evolution of distributed behavior, where evolutionary algorithms are used to discover behaviors for distributed computing systems. We define distributed behavior as that in which groups of individuals must both cooperate in working towards a common goal and coordinate their activities in a harmonious fashion. As such, communication among individuals is necessarily a key component of distributed behavior, and we have identified three classes of distributed behavior that require communication: data-driven behaviors, where semantically meaningful data is transmitted between individuals; temporal behaviors, which are based on the relative timing of individuals' actions; and structural behaviors, which are responsible for maintaining the underlying communication network connecting individuals.

Our results demonstrate that evolutionary algorithms can discover groups of individuals that exhibit each of these different classes of distributed behavior, and that these behaviors can be discovered both in isolation (e.g., evolving a purely data-driven algorithm) and in concert (e.g., evolving an algorithm that includes both data-driven and structural behaviors). As part of this research, we show that evolutionary algorithms can discover novel heuristics for distributed computing, and hint at a new class of distributed algorithm enabled by such studies.

The majority of this research was conducted with the Avida platform for digital evolution, a system that has been proven to aid researchers in understanding the biological process

of evolution by natural selection. For this reason, the results presented in this dissertation provide the foundation for future studies that examine how distributed behaviors evolved in nature. The close relationship between evolutionary biology and evolutionary algorithms thus aids our study of evolving algorithms for the next generation of distributed computing systems.

For Heather.
I couldn't have imagined a better companion for this crazy journey called life. Thank you for your love, support, and encouragement. See you on the beach!

ACKNOWLEDGMENTS

This dissertation would not have been possible without the assistance and support of many:

My deepest gratitude to my advisor, Dr. Philip McKinley. I am sure that I have only just begun to appreciate the value of your guidance.

To Dr. Abdol-Hossein Esfahanian, without whom none of this would have happened: Thank you for giving me a chance.

To the members of my committee, including Dr. Betty H.C. Cheng, Dr. Fred Dyer, Dr. Rich Lenski, and Dr. Charles Ofria: Thank you for your insightful comments on everything from software engineering to fireflies and microbes.

To all the members of the DevoLab that I have had the pleasure of interacting with, especially Charles, Rich, and Jeffrey Barrick : Thank you for opening my eyes to the fantastically odd world of biology!

To Dr. Eric Kasten, whose Friday lunches were a welcome break: Thanks.

To my family, who are perhaps more confused than ever by the Ph.D. process: Thank you for your support.

Thank you to everyone else who has been part of my adventure in graduate school (both times) – You know who you are!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The natural world is replete with communities of organisms that exhibit collective behaviors of varying complexity. For example, nearly all species of microorganism form extracellular structures called biofilms [5], social insects build complex nests [6], ants alert each other of danger through alarm-drumming [7], stickleback fish engage in a simplified form of leader election and school to avoid predators [8], and groups of humans engage in collective decision making [9]. Many of these and other examples include *cooperation*, where individuals work towards a common purpose, as well as *coordination*, where individual efforts are integrated in a harmonious manner. For example, bacteria both cooperate and coordinate to perform *quorum sensing* [10], a mechanism used to trigger a density-dependent change in behavior. Similarly, starlings coordinate via flocking to avoid predation [11]. Finally, although humans may cooperate to make decisions [9], often this process is by no means harmonious!

All of these systems rely upon communication among individuals. This communication may take different forms, e.g., vocalization [12], chemical secretion [10], alarm-drumming [7], dancing (as in the case of honey bees) [13, 14], or stigmergy based on a communally-built

structure (as in termite mounds) [6]. We classify systems that include cooperation, coordination, and communication as instances of *distributed behavior*, where a coherent global behavior emerges out of local interactions among individuals within the environment (c.f. biological self-organization [15]).

Improving our understanding of distributed behaviors has the potential to advance not only the biological sciences, but also to aid in the design of computational systems. In general, the approach taken by our research is to exploit the biological process of evolution by natural selection to discover distributed behaviors for digital systems. It is currently a challenging problem to engineer large-scale distributed computing systems that are resilient in the face of a changing environment [16]. This problem is exacerbated in *cyber-physical* systems, where computing systems interact with the physical world [17]. *Autonomic computing*, so-named for its relation to the autonomic nervous system, was envisioned as a means to address this challenge [18]. Autonomic systems are intended to be fault-tolerant, self-healing, self-organizing, and self-adaptive to their environment (these are known as the *self-*$*$ properties [19]). The technologies underpinning autonomic systems are meant to be general enough to be applied across many different kinds of distributed systems. For example, autonomic systems could be used in large data centers to ease the workload of human administrators [18], or in widely dispersed, hard-to-reach sensor networks [17]. Despite progress in constructing autonomic systems, the state of the practice is still far from the original vision. In contrast, many biological organisms *already* exhibit the *self-*$*$ properties. For example, sensory organs are duplicated for fault-tolerance, wounds heal, communities self-organize, and individuals adapt to their environment.

Parallels between the observed behaviors found in natural systems and the desired behav-

iors of distributed computing systems have not gone unnoticed. Indeed, many bio-inspired approaches to designing distributed systems have focused on *biomimetics*, where behaviors observed in nature are replicated in computing systems [20, 21, 22, 23, 24]. However, while biomimetic approaches have been used to encode various natural behaviors in computational systems, these behaviors are not always well-suited to their new environment. For example, while we may design a crawling micro-robot to behave like an ant, *it is not an ant!* More generally, the *umwelt* [25], or self-centered model of the world, of a robot is different than that of an ant, regardless of how accurately the ant's behaviors are mimicked. Approaches that more directly take into account the capabilities of the target systems, as well as the characteristics of their environment, are needed.

A complementary approach to biomimetics is the evolutionary algorithm, where instead of mimicking the behaviors found in nature, we *harness* evolution by natural selection, the process that originally produced those behaviors. Evolutionary algorithms (EAs) have long been used to discover solutions to complex engineering problems in domains as diverse as antenna design, airfoil shaping, load-bearing structure design, and electronic circuit layout [4, 26, 27, 28]. More recently, an approach known as *digital evolution*, which includes characteristics of both evolutionary algorithms and artificial life, has been used to improve our understanding of evolution in biology. In digital evolution, a population of computer programs are allowed to mutate, replicate, and compete with each other in a common virtual environment (a "digital Petri dish"). Digital evolution has been used to study the evolution of complexity [29] and phenotypic plasticity [30], as well as to improve our understanding of the evolution of group behaviors such as division of labor [31]. The investigations described in this dissertation demonstrate that digital evolution can also be used to help construct

robust distributed systems. Compared to traditional approaches for engineering distributed systems, this approach enables engineers to specify the desired global behavior of the system directly, without *a priori* knowledge of any required local behaviors.

Although the set of all possible distributed behaviors is quite large, we have identified three general classes of distributed behavior observable in both biological and computational systems. First, *data-driven behaviors* are based on the transmission of information between individuals. For example, in a mobile *ad hoc* sensor network, a data-driven behavior could be to calculate the average ambient temperature over all nodes in the network [32]. Second, *temporal behaviors* are based on the relative timing of individual behaviors. An example of this is *heartbeat synchronization*, where all nodes in a network must perform a task at approximately the same time [33]. Finally, *structural behaviors* modify the underlying communication network connecting individuals, and include algorithms for routing [34] and topology maintenance [35]. We note that although these classes of behavior are distinct, most real-world systems include combinations of these behaviors. Moreover, many systems include additional constraints, for example, the speed with which the average temperature is calculated, the maximum jitter between heartbeats, or the fault-tolerance of constructed networks. Importantly, as distributed systems expand into the natural world, an ever-increasing challenge is to engineer systems that not only meet their design objectives, but that also exhibit the *self-*$*$ properties [17]. As our studies show, evolutionary algorithms have the potential to meet these challenges.

While the focus of this dissertation is on the application of evolutionary algorithms to the design of distributed computing systems, the classes of behaviors that we have identified are also seen in biological systems. For example, quorum sensing in a population of bacteria

is coordinated by a chemical called an an autoinducer [10]. In this case, the transmission and reception of the autoinducer is a data-driven behavior, while the coordinated change of behavior is an instance of temporal behavior. A striking example of synchrony in the natural world is the coordinated flashing of male fireflies in some parts of Southeast Asia. In this case, these fireflies synchronize their flashes to a common frequency and phase over a distance of many miles [36], an example of temporal behavior. Moreover, bacterial biofilms contain a complex network connecting bacteria of differing species with tubules that provide nourishment and waste removal [5]. The topology of this network is maintained by the community, a clear example of structural behavior. In addition to their applications in distributed computing, the tools and techniques developed for the research presented in this dissertation enable future computational studies of the associated *natural* behaviors. Computational studies offer many advantages over *in vivo* experiments, including perfect knowledge of behaviors evolved *in silico*.

Thesis Statement. *The integration of evolutionary algorithms and distributed systems enables the discovery of novel and effective distributed behaviors and grants insight into the design of robust distributed computing systems.*

In general, the strategy that we have followed in this research was to develop and/or configure various evolutionary algorithms for application to problems in distributed computing. As the following chapters show, this process has produced *novel* distributed algorithms, and many of the solutions discovered have continued to perform in the presence of network faults, thus remaining *effective*. In the course of our research, the following contributions have been made in support of this thesis:

1. We have proposed and evaluated evolutionary algorithms that enable the evolution of

distributed behaviors.

2. We have demonstrated *de novo* evolution of three different classes of distributed behaviors, including data-driven, temporal, and structural behaviors.

3. We have demonstrated the evolution of compound systems comprising multiple classes of distributed behaviors.

4. We have developed tools and techniques for computational studies of evolutionary processes, specifically those related to group-level behaviors such as cooperation and coordination.

The remainder of this dissertation is organized as follows. Chapter 2 reviews related work and describes the Avida digital evolution platform, including our main extensions to Avida that enabled the studies described in this dissertation. In Chapter 3, we describe our study of data-driven behavior, which is focused on the evolution of consensus, a behavior whereby agents must agree on a course of action. This chapter also includes an investigation of the effect of population structure on the evolution of consensus. In Chapter 4, we examine the dual problems of synchronization and desynchronization, both of which are examples of temporal behavior. In Chapter 5, we address the construction of communication networks, an important characteristic that underlies all distributed behaviors. In each of these chapters, we also provide specific background and related work, and describe extensions to Avida that were required for that study. In Chapter 6, we explore the evolution of a compound behavior, where we require the simultaneous evolution of multiple kinds of distributed behavior for a single problem, specifically the area covered by a simulated mobile *ad hoc* network. Finally, Chapter 7 concludes and describes potential future work.

Chapter 2

Background and Related Work

In this chapter we review related work and provide background information on relevant topics: bio-inspired distributed computing, evolutionary robotics, and the evolution of communication and cooperation. We also briefly describe evolutionary algorithms, and conclude this chapter with a description of the Avida platform for digital evolution, the software system employed for the majority of our research, focusing on our extensions to support this research.

Distributed behaviors are of interest to two overlapping branches of engineering: distributed problem solving and distributed control. Traditionally a subfield of distributed artificial intelligence, *distributed problem solving* is the use of multiple, semi-autonomous, and cooperating software agents to solve a problem [37]. Examples include establishing a distributed sensor network within the environment, and then using that network for monitoring vehicle movements [37]. *Distributed control*, also known as *cooperative control*, is primarily concerned with the coordinated control of multiple devices, such as the stages of an assembly line [38] or a swarm of robots [39]. These two areas are collectively referred to

as *Multi-Agent Systems* (MAS) [37].

Bio-inspired approaches to developing multi-agent systems tend to fall into two broad categories: biomimetic approaches, where behaviors are based on observable phenomena found in nature; and approaches where behaviors are discovered by evolutionary algorithms. Let us consider each in turn.

2.1 Biomimetic Distributed Behaviors

A *biomimetic* system contains some feature or property from a natural system that has been found to be beneficial in distributed computing. A key aspect of many distributed systems is that *global* behaviors are the result of the collective actions of individuals operating on *local* information. In biology, this characteristic is known *self-organization* [15]. In computing, a related concept is *emergence*, where the behavior of individual agents within a system are engineered in such a way as to give rise to a desired global behavior [22]. The close relationship between self-organization and emergence suggests that self-organizing biological systems can inspire approaches to engineering emergent biomimetic systems. For example, the slime mold *Physarum polycephalum* has been shown to form networks that are as efficient and fault-tolerant as real-world transportation networks [40]. In addition, the flocking behavior of migratory birds has inspired emergent algorithms for leader-election [41], and observations of the social behavior of insect colonies have inspired a variety of approaches to controlling swarms of mobile robots [21, 22, 23]. Emergence can sometimes be characterized by a reduction in information entropy [42, 43], however, there are few guidelines for how one develops an emergent system, and the general principles that underly such systems remain unclear.

One area in which biomimetic approaches have enjoyed considerable success is in the construction and maintenance of networks [24, 40], an important component of all distributed systems. Of course, changing the infrastructure of large-scale networks is costly and time-consuming, and thus many modern distributed systems rely on *overlay networks*, which are communication networks maintained by application software, built on the underlying physical topology [35]. Overlay networks can be thought of as "logical," or virtual, networks that are built using components of a single physical network. Figure 2.1 depicts two different overlay networks built atop the same physical network. As can be seen here, overlay networks can vary in structure even as the physical topology remains constant. This design makes overlay topology management an ideal candidate for bio-inspired methods. For example, Snyder et al. [44] demonstrated the effectiveness of an overlay network whose structure was based on the growth pattern of fungal hyphae. In a different study, Jelasity et al. [45] described an approach to constructing overlay networks based on simple gossip-based interactions among nodes. These studies were the inspiration for our own research into the evolution of network construction, described in more detail in Chapter 5.

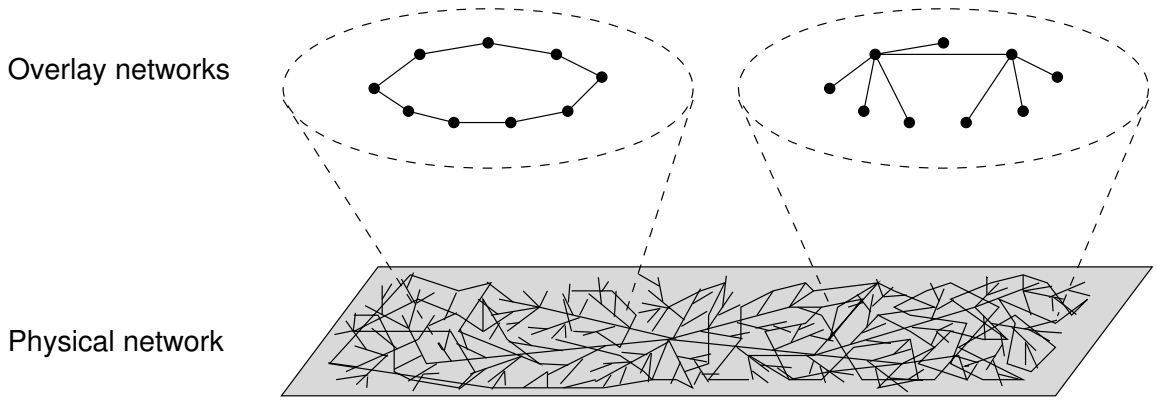


Figure 2.1: Two overlay networks built atop an underlying physical network: A distributed hash table, where each node in the network stores multiple key-value pairs, and the structure of the overlay network is used to route search requests [1] (left); and a tree-structured peer-to-peer overlay network that provides rapid data dissemination [2] (right).

Other researchers have proposed biomimetic approaches to designing temporal behaviors. For example, a common problem in distributed systems is *heartbeat synchronization*, where agents within a network must perform a task at approximately the same time. Classical approaches to synchronization include a wide array of consensus and scheduling algorithms [46], and such algorithms are a common part of protocols that require restarts [45]. Biomimetic approaches to synchronization include the work of Babaolgu et al. [33], whose research was inspired by models of firefly synchronization proposed by Ermentrout [47] and the mathematical modeling of pulse-coupled oscillators by Mirollo and Strogatz [48]. In a related study, Patel et al. [49] described two different protocols for *de*-synchronization, one based on particle diffusion and another based on firefly synchronization. These two studies were the inspiration for our own investigation into the evolution of temporal behaviors, described in more detail in Chapter 4.

2.2 Evolutionary Algorithms

While biomimetic systems are effective at translating natural behaviors to a digital world, they are limited to the behaviors observable *today*. In contrast, *evolutionary algorithms* (EAs) encode the biological process of evolution by natural selection in software so that they can be used to search for solutions to complex, high-dimensional problems [50]. For example, the well-known genetic algorithm has been used to solve a wide-variety of engineering optimization problems [51], while genetic programming has consistently produced human-competitive results in fields as diverse as circuit layout and antenna design [52, 3].

One of the main advantages of EAs compared to traditional engineering approaches is that they are not limited or constrained by human preconceptions about how a given

Figure 2.2: Two antennas developed at NASA, one by human engineers (left), and another discovered by an evolutionary algorithm (right). The evolved antenna was scheduled for deployment on satellites for NASA's Space Technology 5 Mission. Figures from [3] and [4], respectively. For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

problem should be solved. For example, Figure 2.2 depicts two antennas, one engineered by humans, and another discovered by an EA. In this case, not only was the evolved antenna fully compliant with design objectives, it was smaller, easier to manufacture, used fewer components, and consumed less power than the human-engineered antenna [3].

Generally speaking, most evolutionary algorithms follow a similar process, outlined in Figure 2.3. For example, let us assume that we wished to evolve a control algorithm for the nodes in a mobile *ad hoc* network (MANET) using HyperNEAT [53], an EA that produces artificial neural networks. We begin by generating a population of random neural networks. Next, we evaluate each neural network using a *fitness function*, a user-defined function that indicates how well a given neural network solves our problem. When using an EA for control algorithms, it is common for this fitness function to be based on the behavior of the controller in a simulated environment. In this case, let us use a simulation environment like that shown in Figure 2.4(a), and define fitness as: "The number of unique grid cells occupied by the largest connected component of the network." Through a repeated process of fitness

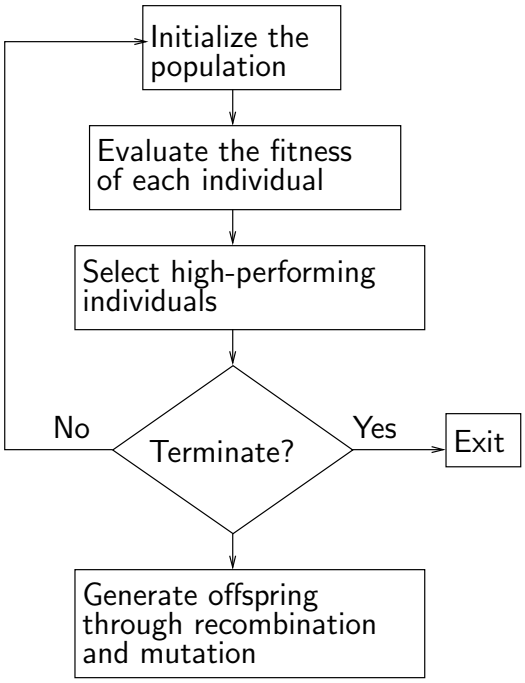
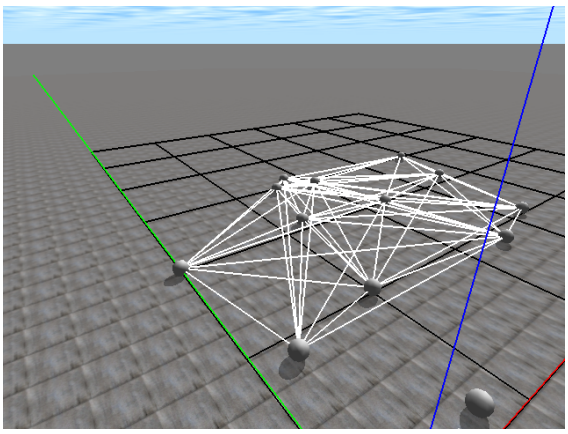


Figure 2.3: Flowchart depicting the process followed by many evolutionary algorithms.

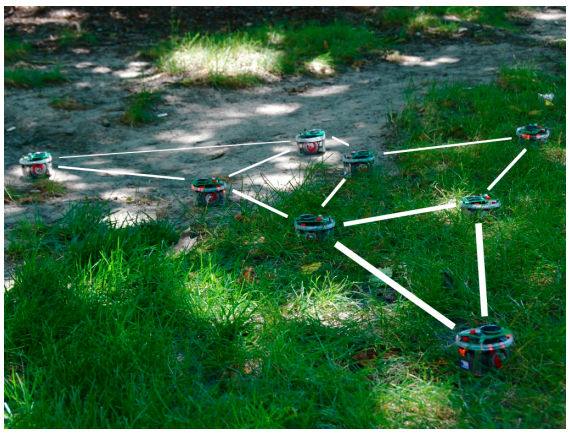
evaluation, mutation, and selection, this evolutionary algorithm will discover neural networks that control the MANET to solve this task. Ultimately, the evolved control algorithm would then be deployed on a real MANET, like that shown in Figure 2.4(b). (This scenario is in fact drawn from our own research, and will be described in more detail in Chapter 6.)

2.2.1 Evolutionary Algorithms in Biology

Because evolutionary algorithms are based on the biological process of evolution by natural selection, they can frequently be used to help us understand evolution itself [55]. A quote by famed evolutionary biologist John Maynard Smith succinctly captures the motivation for research in computational evolutionary biology: "So far, we have been able to study only one evolving system and we cannot wait for interstellar flight to provide us with a second. If we want to discover generalizations about evolving systems, we will have to look at artificial



(a) Simulated mobile ad hoc network. Nodes are depicted as spheres, white lines represent network connections between nodes. Image from [54].



(b) Mobile ad hoc network comprising 8 E-Puck robots http://www.e-puck.org/. White lines again represent connections between nodes.

Figure 2.4: Example simulation environment for a mobile ad hoc network (a) and corresponding physical system (b).

ones [56]."

Due to the complex nature of evolution and the scale required (thousands of individuals, hundreds of generations), long-term studies of evolutionary processes in natural organisms, even relatively fast replicating bacteria, require extensive commitment of time and resources [57]. Evolutionary computation offers a complementary approach to uncover generalizations about the process of evolution. For example, understanding the evolutionary basis of cooperation and communication is a long-standing question in biology [58]. Numerous biological hypotheses have been proposed, including group-focused hypotheses such as multilevel selection [59, 60, 61] and partner fidelity [62], as well as gene-focused hypotheses such as the selfish gene and extended phenotype [63, 64]. To better understand the evolution of these behaviors, computational studies have employed both game-theoretic and evolutionary algorithms [58, 65, 66, 67]. These approaches have helped to describe the conditions under which cooperation might evolve [68,69], and have also investigated the evolution of communi-

cation [70, 71, 72, 73, 74]. Finally, digital evolution experiments with the AVIDA [75] platform have contributed to our understanding of the evolution of complex features, altruism, and division of labor [29, 76, 31].

2.2.2 Evolutionary Algorithms for Distributed Behavior

In addition to addressing the theoretical underpinnings of evolution in natural organisms, evolutionary algorithms can also be used to study distributed behaviors. Here, EAs can be used to discover previously unknown solutions to engineering problems. Different EAs, such as genetic algorithms [77, 51], genetic programming [52], neuroevolution [78], and digital evolution [79] codify the process of evolution by natural selection in different ways. These techniques have been applied to a variety of problems in distributed computing, including multicast mapping [80], multi-agent systems [81], and the automated design of communication protocols [82]. Hybrid EAs that use genetic programming to influence swarm dynamics have also been proposed [83].

Coordinating the behavior of multiple agents is a common problem in evolutionary robotics, where evolutionary algorithms are used to evolve controllers for (potentially different) robots [84, 85, 66]. For example, a single robotic controller for complex object manipulation can be evolved [86], or a team of robotic controllers can be evolved to forage for resources [87] or cooperate in some other collective action [88]. Some applications of evolutionary robotics, specifically those involving teams [88], have also granted insight into how cooperation might evolve [66,85]. In addition to evolving control algorithms, researchers have also co-evolved controllers concurrently with the robot's morphology [89,86]. Compared to traditional approaches to robotics, evolutionary approaches promise to discover behaviors

that are tuned specifically for the robot and task at hand.

Our research is related to evolutionary robotics insofar as both have been used to evolve group behaviors. Indeed, as will be shown in Section 6, there are clear extensions to our research that cross over into evolutionary robotics. However, our research starts from the evolution of distributed *logical* behaviors, while evolutionary robotics starts from the evolution of *embodied* behaviors. While evolutionary robotics is intrinsically robotics-oriented, our research is applicable to a broad class of distributed behaviors that are disassociated from a physical presence.

To conduct this research requires an evolutionary platform that can capture complex environments and different selective pressures, while remaining abstracted from the constraints of the physical world. In the next section, we discuss AVIDA [75], a platform for digital evolution. AVIDA is the most-well known approach for computational evolutionary biology, and the platform that we have used for the majority of research described in this dissertation.

2.3 Avida: A Platform for Digital Evolution

In digital evolution [79], a population of computer programs exists in a user-defined computational environment. These "digital organisms" self-replicate, compete for available resources, and are subject to instruction-level mutations and natural selection. Over thousands of generations, they can evolve to survive, and even thrive, under extremely dynamic and adverse conditions. Unlike biomimetic approaches, digital evolution is not confined to behaviors found in existing living organisms. Moreover, digital evolution is open-ended: whether a given organism self-replicates and moves into the next generation depends on its environment and its interaction with other organisms.

AVIDA [75], a platform for digital evolution, has been used to study the evolution of biocomplexity in nature [90, 29] and to address complex problems in science and engineering [91,92,93], including the construction of communication networks [94] and energy-efficient population control [95]. In the course of our research, we have extended AVIDA in several ways to support the evolution of distributed behaviors. Specifically, we have provided digital organisms in AVIDA with several rudimentary communication mechanisms, comparable to the capabilities of nodes in a sensor network. We have also enabled the evolution of group behaviors in AVIDA through multilevel selection, which will be discussed in more detail below. In general, our approach has been to use AVIDA as a platform to investigate evolutionary dynamics. As will be seen in Chapter 6, we have also applied knowledge gained from our AVIDA experiments to other evolutionary systems.

2.3.1 Basic Avida Operation

Let us begin with a brief overview of AVIDA operation; additional details may be found in [75]. Figure 2.5 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. Instructions in AVIDA can also have different costs (in terms of virtual CPU cycles) associated with them. For example, a simple addition instruc-

tion may cost only one cycle, while broadcasting a message may cost 20 cycles. Different AVIDA virtual CPU architectures have been implemented and used in various studies. The virtual CPU architecture used in our studies contains a circular list of three general-purpose registers $\{AX, BX, CX\}$ and two general-purpose stacks $\{GS, LS\}$, and four special-purpose *heads* (next instruction, flow control, read, and write), which may be thought of as pointers into the organism's genome.

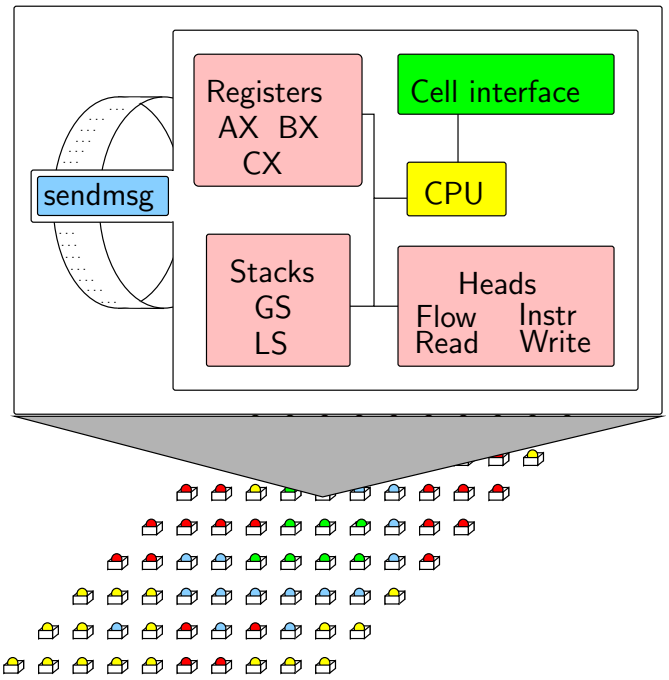


Figure 2.5: An AVIDA population, where different colors represented different genomes (bottom), and the structure of an individual organism (top).

As shown in Figure 2.5, each organism in AVIDA lives in a *cell* located in a fixed location in a spatial environment (cells can contain only a single organism; organisms cannot live outside of cells). Each cell in the environment has a circular list of directed *connections* to neighboring cells, and these connections define the topology of the environment. 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). Each

organism also has a *facing*, selected from its cell's connections, that defines its orientation. Cells can also hold information, which we call *cell data*. This data, which is simply a 32-bit integer, can be used in a variety of different ways. For example, if cell data is randomly assigned, individuals can be tasked with discovering the largest value present in the environment. The virtual CPU instruction collect-cell-data can be used by individuals to access the cell data at their location in the population.

On average, each digital organism is allotted a configurable number of virtual CPU cycles per update, where an *update* is the standard unit of time in Avida. However, during an Avida experiment, the *merit* of a given digital organism determines how many instructions its virtual CPU is allowed to execute relative to other organisms. As opposed to more traditional EAs, where candidate solutions do not share a common environment, digital organisms are self-replicating. Thus, organisms with a higher merit (all else being equal) replicate more frequently, spreading throughout and eventually dominating the population. Moreover, merit in Avida is updated asynchronously based upon performed *tasks*. Tasks are designed by the user and are used to reward desirable behavior (they may also punish undesirable behavior), thereby driving natural selection. Tasks are defined in terms of an organism's phenotype, e.g., the content of messages sent to neighboring organisms, rather than in terms of the specific virtual CPU instructions executed during an organism's lifetime. This approach is intended to allow maximum flexibility in the evolution of a solution for a particular task.

During self-replication, digital organisms allocate memory for their offspring's genome, copy instructions from their genome to their offspring's, and finally issue a divide instruction, which causes the offspring to be "born." An Avida population typically starts with a single

self-replicating organism, and different genomes are produced through random mutations during self-replication. These mutations may take the form of a replacement (substituting a random instruction for the one copied), an insertion (inserting an additional, random instruction into the offspring's genome), or a deletion (removing the copied instruction from the offspring's genome). When an organism replicates, a target cell that will house the new organism is selected from the environment. Different models to select this target cell are available, including mass-action (select at random from among all cells) and neighborhood (select from cells adjacent to the parent), among others. In every case, an organism that is already present in the target cell is *replaced* (killed and overwritten) by the offspring.

In Avida, the *default ancestor* is the organism that begins each experiment. This organism's genome typically contains the instructions needed for self-replication, as well as a large number of nop-C instructions, which perform no useful 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. Many of our studies used a default ancestor that was not capable of self-replication, specifically those in which case organisms were replicated as part of deme replication. This process is described in more detail below.

2.3.2 Communication Among Digital Organisms

To support our research into the evolution of distributed behavior, we extended AVIDA with a series of virtual CPU instructions that enable digital organisms to communicate with each other via message-passing. These instructions are summarized in Table 2.1. Communication among digital organisms proceeds as follows. First, the sending organism must execute a send-msg instruction, which marshals two four-byte values (the *label* and *data* fields, respectively) into a message and sends the message in the direction currently faced. If the sending organism is facing a neighboring organism, the message is deposited in that neighbor's receive buffer. If the sender was facing an empty cell, the message is lost. The rotate-left-one and rotate-right-one instructions can be used to alter the sender's facing. The recipient of the message must then execute a retrieve-msg instruction to extract the contents of the most-recently received message from its buffer and place the two message fields in registers of its own virtual CPU. In addition to this basic form of unicast communication, we also provide broadcast capabilities via the bcast1 instruction. When this instruction is executed, a message is marshaled as described above and copies of this message are sent to each neighboring organism. Recipients of a broadcast message may then execute the retrieve-msg instruction to access the contents of this message, as described above.

These five instructions comprise the basic message-passing capabilities of digital organisms in AVIDA. We note that organisms are not automatically able to determine if they are facing an occupied cell, nor do we provide an explicit mechanism for them to identify neighbors (though these capabilities may be evolved). We also place no restrictions on the semantics of message fields, thus providing evolution with maximum flexibility to discover novel behavior. In general, all of the AVIDA studies described in this dissertation included

Table 2.1: Communication instructions used with AVIDA for evolving distributed behavior. All instructions are equally likely to be selected as targets for mutation. As with many AVIDA instructions, these instructions are *NOP-modifiable*, where trailing nop-∗ instructions alter the specific virtual CPU registers that are used.

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.
bcast1	Sends a two-word message containing the values of registers BX and CX to all immediately neighboring organisms.
rotate-left-one	Rotates the caller counter-clockwise one step.
rotate-right-one	Rotates the caller clockwise one step.
collect-cell-data	Loads the caller's BX register from the data held by the caller's cell.

these instructions, and in some cases additional communication-related instructions were provided; these are described in the methods for the appropriate study. Cell data (mentioned above) is commonly used in conjunction with communication to provide individuals with a unique identifier; the collect-cell-data instruction is also shown in Table 2.1.

2.3.3 Multilevel Selection

The theory of group selection was originally proposed in 1962 by the biologist V. C. Wynne-Edwards [96], and later formalized into multilevel selection theory by D. S. Wilson and E. Sober [97, 61, 98]. At a high level, multilevel selection theory states that groups of individuals may be vehicles for selection, similar to how a single individual is a vehicle for the selection of its genes. In other words, multilevel selection posits that the survival of the individual is *linked* to the survival of the group. There are many different ways that these groups may be defined. For example, a group may be defined by a common trait (a trait-group), shared ancestry (clade selection), membership in the same species (species selection), or the interactions between related individuals (kin selection). Multilevel selection has been used

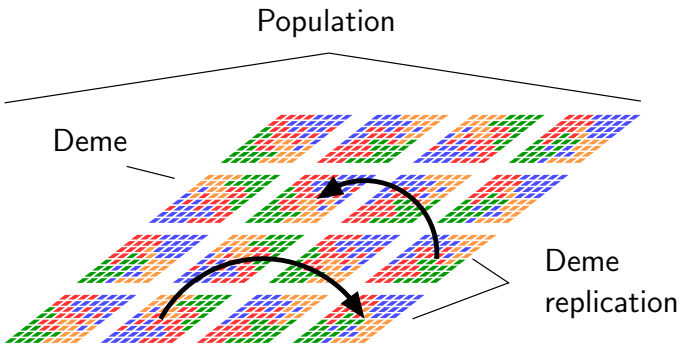


Figure 2.6: Depiction of an AVIDA population of sixteen demes. Demes are isolated subpopulations, each capable of replication. When a deme replicates, it replaces a randomly selected target deme.

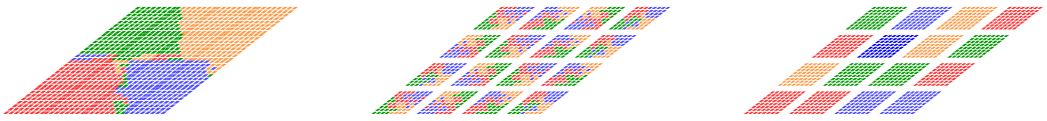
in biology to explain the evolution of altruism, where an individual will sacrifice fitness for the benefit of the group, and the evolution of social behavior, particularly in populations of social insects, such as ant and bee colonies [99, 100, 101, 102].

In AVIDA, the key difference between individual selection and multilevel selection experiments is the formation of groups within the AVIDA population. Typically, groups are formed by splitting the population into isolated subpopulations, called *demes*; each deme is one such group. Figure 2.6 depicts an environment that has been subdivided into sixteen demes. As with individual organisms, demes also replicate and compete with each other for space. When a deme (the *source*) replicates, another deme from the population is selected for replacement (the *target*). The organisms in the target deme are removed, and a subset of the organisms from the source is cloned and placed into the target deme. In Figure 2.6, we show two demes replicating – Organisms in the target demes will be removed from the population, and these target demes will be repopulated from their respective source demes.

Deme replication can be triggered in two different ways. For example, deme replication can be triggered asynchronously as a result of the deme's behavior, a process which is supported by the ReplicateDemes framework. In contrast, the CompeteDemes framework enables

the periodic replication and competition of demes. Deme replication can also be combined with a deme-level merit, where all digital organisms within the deme receive additional merit based on the behavior of their deme. Similar to the individual merit described earlier, a deme merit increases the number of virtual CPU instructions its inhabitants are allowed to execute, relative to other demes in the population. Use of either deme merit, ReplicateDemes, or CompeteDemes thus provide a selective pressure that operates on entire demes.

The introduction of demes to AVIDA has enabled studies that span multiple levels of selection, some of which are depicted in Figure 2.7. In this figure, each color represents a different genotype. In Figure 2.7(a) we see individual selection only, where organisms compete with each other for space (cells) in their environment and are responsible for their own self-replication. In Figure 2.7(b), we see multilevel selection, where both individuals and demes replicate. In this case, demes are heterogeneous, and organisms not only compete with each other for space *within* each deme, but also *among* demes.



(a) Individual selection, single population.

(b) Multilevel selection, heterogeneous demes.

(c) Group selection, homogeneous demes.

Figure 2.7: Different kinds of common multilevel selection experiments supported by AVIDA. Different shades represent different genotypes.

We have extended AVIDA to support group selection[1] on homogeneous demes, depicted in Figure 2.7(c). Here, the population is again split into demes, however, organisms within each deme are clonal. In this case, a heritable lineage of genomes, called a *digital germline*

[1]Although there exists some controversy regarding the role of group selection in natural evolution, researchers such as Waibel, Keller, and Floreano [103] have demonstrated its effectiveness for evolving collective behaviors of artificial systems.

is attached to each deme, and all organisms within a deme are instantiations of the latest genome from their respective germline [94]. Digital germlines in AVIDA play a role similar to that of germlines in animals, where genetic material is transferred from parent to offspring along the germline [104]. When a deme with a germline replicates, any mutations occur to the deme's germline. This process, illustrated in Figure 2.8, is called GermlineReplication, and proceeds as follows. Beginning from an ancestral germline g_0, the "parent" deme is seeded with an organism generated from the latest genome along the germline. During the course of the experiment, the organisms within this deme replicate, compete for resources, and may experience mutations. Once a deme replication is triggered, all organisms within the parent deme are killed, and the parent is re-seeded from its germline, g_0. The latest genome from g_0 is then mutated, producing a new germline, g_1. An "offspring" deme is randomly selected from the population, any organisms currently living in this deme are killed, and the new genome from g_1 is used to seed the offspring deme. As in the group selection case, replication when using a germline can be either synchronous or asynchronous.

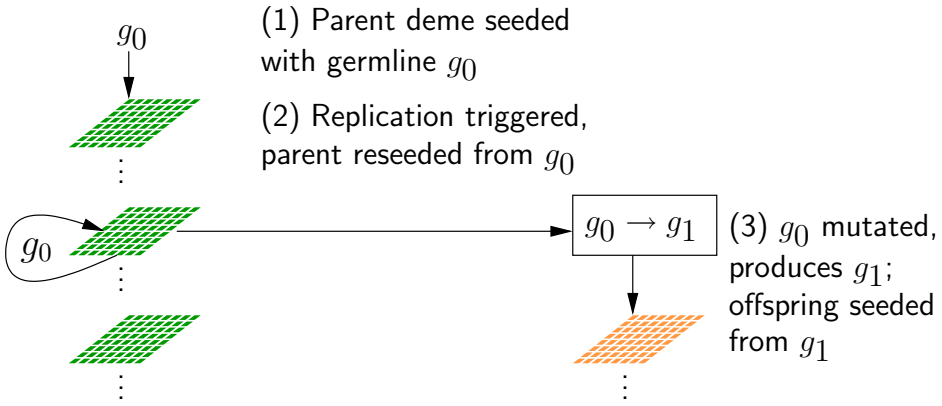


Figure 2.8: Example of GermlineReplication.

For many of the studies described later, we used the CompeteDemes framework in combination with a digital germline. 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 selection on the demes, preferentially replicating those demes with higher fitness, and replacing those demes with lower fitness. For example, we may define fitness functions based on the degree to which organisms within a deme perform a data-driven behavior. Then, over time, the CompeteDemes framework will preferentially replicate those demes that perform the requested task, 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. In the next chapter, we describe such a study of data-driven behavior, distributed consensus.

Chapter 3

Evolving Data-driven Behavior: Consensus

Data-driven behaviors, or those behaviors where data sent between individuals is semantically meaningful, are prevalent in biological systems that employ communication. The alarm calls of prairie dogs [105], stigmergic communication among ants [7], and the waggle dance of honey bees [13, 14] are all examples of behaviors that convey information among individuals. Some of these cooperative behaviors include *consensus*, where members of a group agree upon a particular course of action. An example is the schooling behavior of stickleback fish, where the school "agrees" to follow one (or a few) "leader" fish [8]. In this chapter, we describe our studies into the evolution of consensus in digital systems.

3.1 The Consensus Problem

In computing, the *consensus problem* is where a group of agents are required to reach agreement. The agents may be any combination of hardware or software, from computer processes

in a network to mobile robots. Likewise, "agreement" can vary from calculating the average temperature of a monitored environment in the case of a mobile network, to selecting a single process from a pool to perform a database operation.

Although simple to understand, bounded-time consensus in the presence of failures has, in fact, been proven impossible (the FLP impossibility proof) [106]. This result showed that in systems relying on message-passing, one can always construct a series of operations that will prevent consensus from being reached in bounded time. However, *in practice*, heuristics for consensus are quite effective [46, 107]. While the motivation for our work stems from consensus in distributed systems, consensus is studied in fields as diverse as coordination games [108] and cooperative control [38]. As mentioned earlier, coordinating the behavior of multiple agents is also a common problem in evolutionary robotics [84], and artificial life studies have contributed to our understanding of the evolution of cooperation and communication [67], all of which require some degree of consensus among agents.

In distributed computing systems, consensus algorithms are frequently used to ensure consistency between replicated components in an effort to increase reliability [109]. In practice, algorithms such as Paxos [110] are used as the basis for implementations of consensus [111], which in turn can support higher-level services such as distributed lock management [112]. However, as computing systems continue to expand their reach into the natural world, for example through cyber-physical systems [17], 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. One such complication is the shift from systems comprised of few high-performance nodes to systems containing large numbers of unreliable, commodity systems [113].

Our studies: In the remainder of this chapter we describe a series of experiments we conducted to evolve consensus in Avida. We start with leader election, a specific form of consensus where digital organisms are required to "elect" one of their number as a leader. We describe two different approaches to leader election, one that used only individual selection, and another that used multilevel selection. We then describe our studies of a general form of consensus, and show how Avida is able to discover novel heuristics for consenus. Finally, using this same general form of consensus, we investigate the impact of various types of biologically-inspired population structures on the evolution of consensus.

3.2 Directed Evolution of Leader Election

Our first study into the evolution of consensus focused on guiding an Avida population to perform *leader election*, a distributed problem in which the population as a whole must "elect" a single individual to perform some task [46]. Here we use a single population (that is, the population is *not* divided into demes) and use Avida tasks to provide individual-level selection pressures. In this study, we assigned each cell a random number (via cell data), and defined leader election as determining the largest cell data sensed by any individual in the population, similar to how a sensor network might detect events of interest [114]. Although this was a highly constrained problem, its utility was to improve our understanding of the challenges to be faced in evolving collective behavior in Avida. In the remainder of this section, we describe our methods and results in detail, and conclude with an analysis of the dominant genome that displayed the desired behavior.

Methods. As noted, we assign all cells random cell data. For brevity, we refer to this cell data as the identifier (ID) of the organism occupying that cell. Thus, the desired behavior is that all individuals in the population send messages containing the largest ID. Taking into account population turnover and mutations, we consider the solution to have been found - that is, leader election has occurred - when 95% of messages carry the largest ID.

We conducted three different sets of experiments. Each uses different combinations of AVIDA tasks, however, all are focused on evolving the same behavior: proliferation of messages that carry the largest sensed value. The first set of experiments investigates the basic communication capabilities of digital organisms, focusing on the evolution of a message filtering behavior. The second introduces a penalty, where each time that a digital organism sends a message that does not contain a valid ID, the sender's merit is reduced. Finally, the third set of experiments investigates the ability of the population to recover from resetting the largest ID. We note that organisms do not have an inherent ability to identify messages that contain IDs; both the messaging behavior, as well as the grammar, must be evolved.

For this study we configured AVIDA as follows. The environment comprises 3600 cells in a 60×60 torus. Experiments are run for 100,000 updates, where an *update* is the standard unit of time in AVIDA trials, and corresponds to the execution of 30 instructions per virtual CPU, on average. The copy mutation rate is set to 0.75% per-instruction, while the insertion and deletion mutation rates are set to 5% per-replication; these parameters correspond to the default AVIDA configuration. We developed a set of tasks, summarized in Table 3.1, to reward organisms for various communication behaviors. The send-self task rewarded organisms for sending messages that contained their own ID, while the send-id task rewarded organisms for sending messages that contained *any* ID. The max-known task rewarded organism for sending

Table 3.1: Descriptions of the AVIDA tasks developed for the directed evolution of leader election study.

Task Name	Description
send-self	Rewards sending a message containing the sender's ID.
send-id	Rewards sending a message containing *any* ID.
max-known	Rewards sending a message containing the largest value known, defined as $Max(self, Max(msg_0, ...msg_n))$, where $\{msg_0, ...msg_n\}$ is the set of all messages received by that organism. The sender must have received at least one message prior to being rewarded for performing this task.
send-non-id	Penalizes the sender of a message that does not carry an ID.

a message that contained the maximum ID they had ever received in a message. Finally, the send-non-id task penalized (via a negative reward) the sender of a message that did not contain an ID. Various combinations of these tasks are used in the following experiments. All communication and cell data instructions described in Section 2.3.2 (except broadcast) were made available, and 20 separate AVIDA trials were performed for each experiment.

3.2.1 Message Filtering

In the first experiment, we tested the hypothesis that rewarding organisms for sending messages containing IDs, where the ID carried is greater than the organism's own ID, will eventually result in all messages in the population carrying the largest ID. Experiments were conducted using different combinations of the tasks defined in Table 3.1. In every case, experiments that used the max-known task produced organisms that sent messages containing IDs greater than their own. However, none of these experiments resulted in the proliferation of the largest ID.

Figure 3.1 depicts average messaging behavior for an experiment that uses the max-known and send-id tasks. Five different values are plotted every 100 updates: Total, the

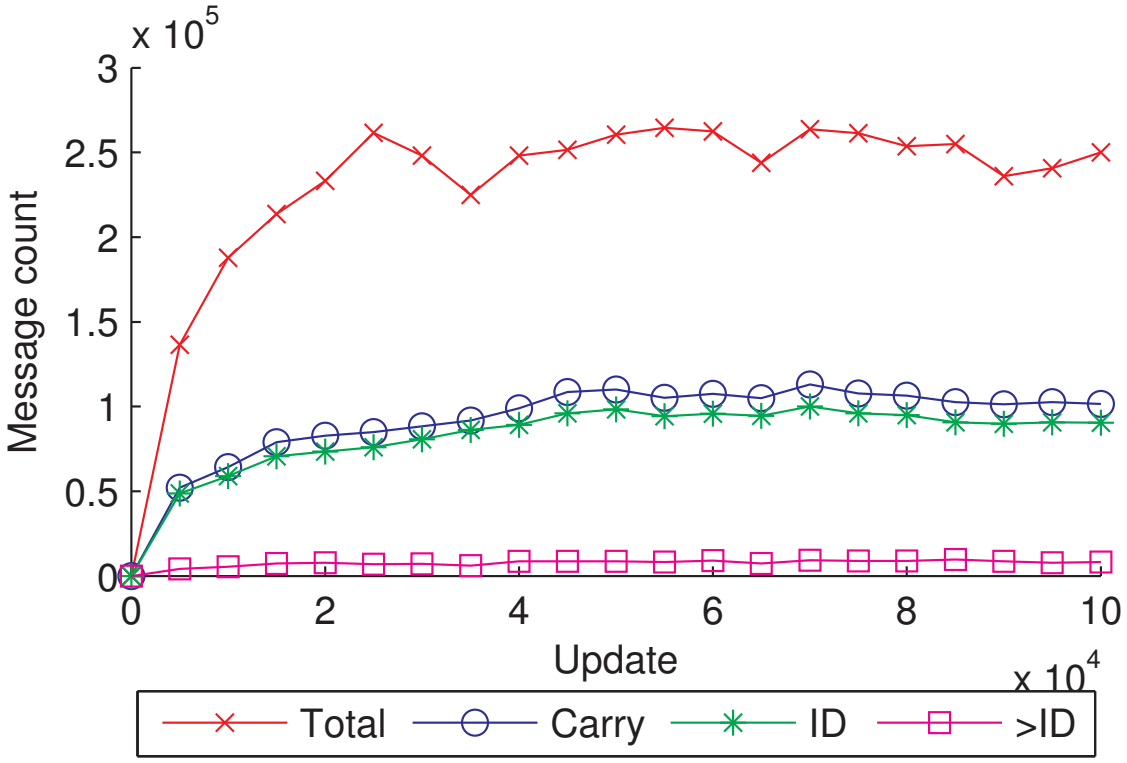


Figure 3.1: Average messaging behavior when using the max-known and send-id tasks.

total number of messages sent; Carry, the number of messages sent that carry an ID; ID, the number of messages sent that carry the sender's ID; and >ID, the number of messages sent that carry a ID greater than the sender's. Here we see that more than half of all messages sent do not carry an ID, indicated by the difference between Total and Carry. Effectively, these are "junk" messages, produced when organisms send values that are easy to calculate or when a send-msg instruction has recently been mutated into the genome. We also see that greater than 75% of ID-carrying messages contain the sender's ID. Finally, very few messages contain an ID that is greater than the sender's ID.

Figure 3.2 shows the average number of *organisms* that performed the max-known and send-id tasks during the same Avida trials. Here we see that when max-known task is used in combination with the send-id task, not only do all (allowing for genetic drift) organisms

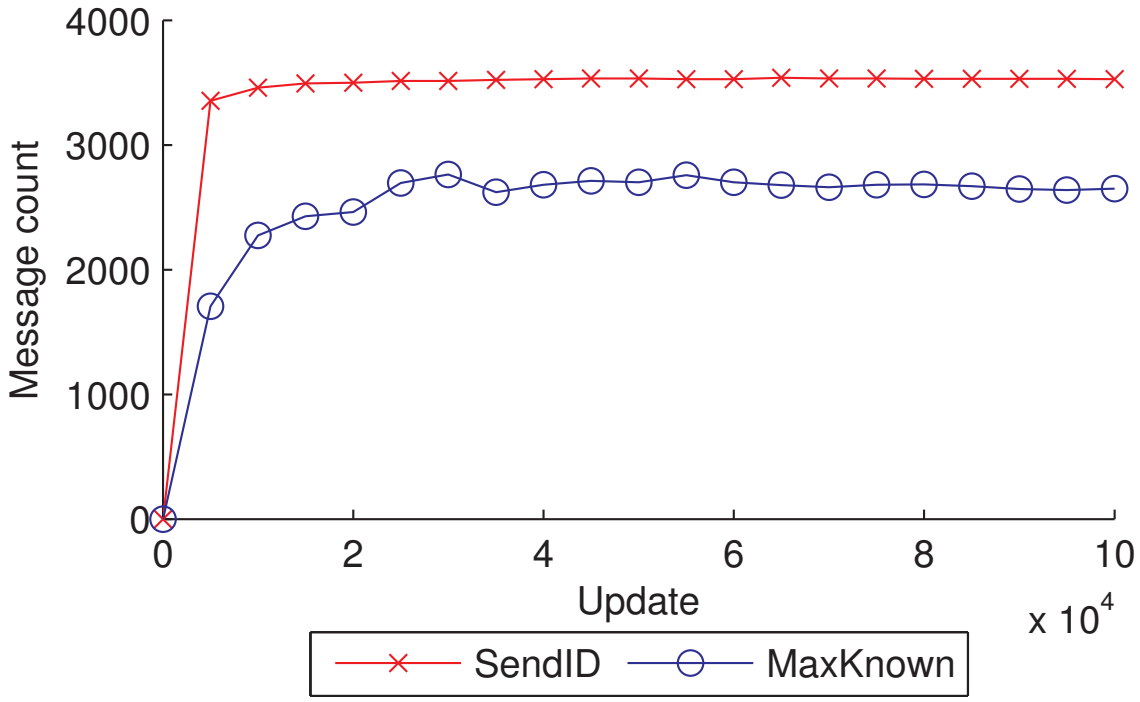


Figure 3.2: Average task performance when using the max-known and send-id tasks.

perform the send-id task, but a large number of organisms (2600) also perform the max-known task. We note that organisms can perform the max-known task by sending their own ID once they have received any message carrying a smaller value.

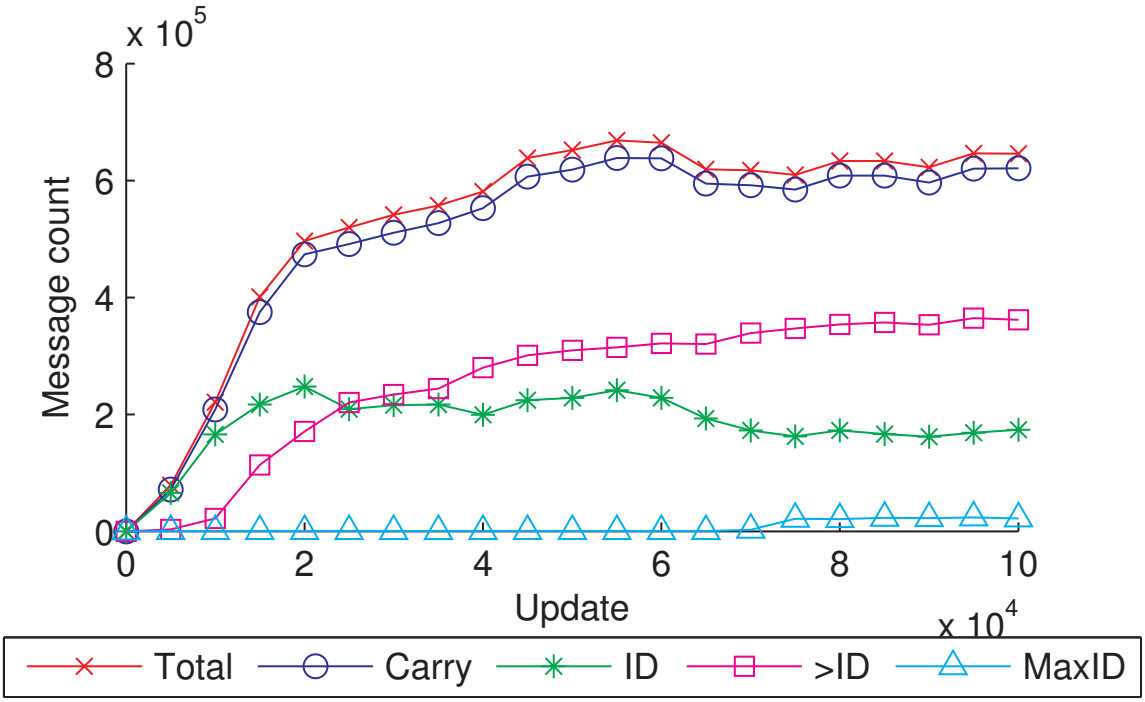
3.2.2 Encouraging ID-Carrying Messages

Our next experiments investigated ways to reduce the number of junk messages being sent, under the supposition that the large number of non-ID carrying messages might be preventing the population from determining the largest ID. We tried two different approaches, one where we actively penalized organisms for sending junk messages, and another where we increased the cost (in virtual CPU cycles) of the send-msg instruction. Both of these approaches resulted in the desired behavior, with the penalty evolving a solution more quickly than the additional cost approach (data not shown).

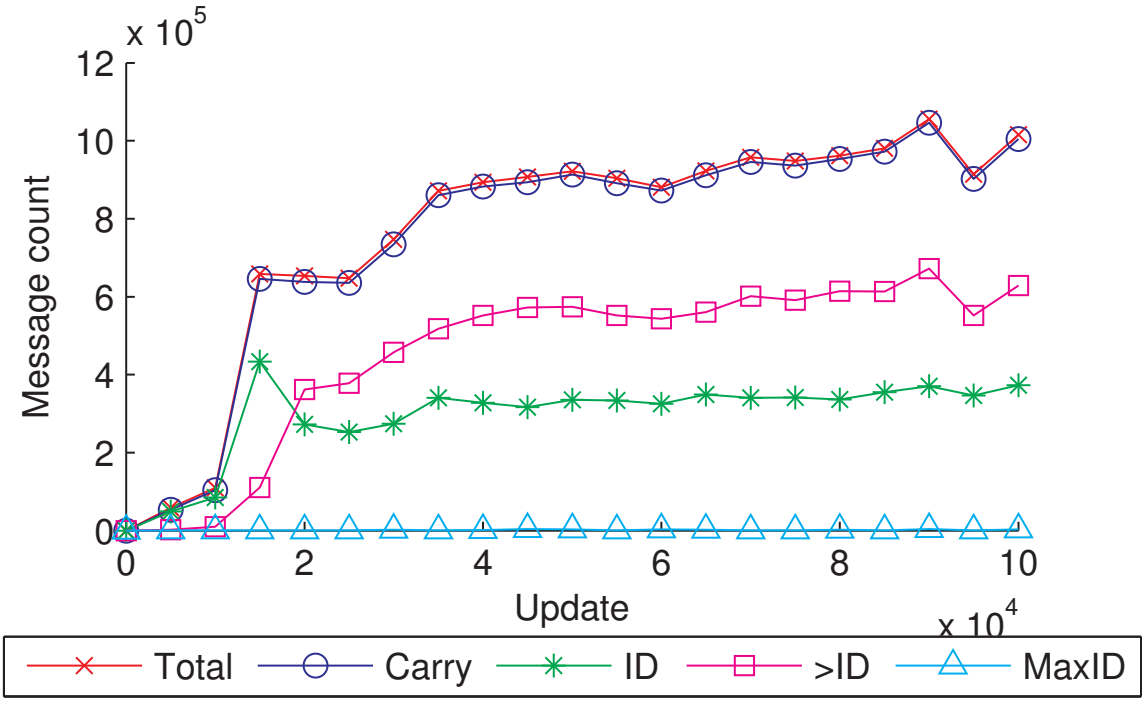
In this experiment, a task, send-non-id, was defined such that the sender of a message that does not carry a cell-ID is docked 75% of its merit. The send-non-id task is similar to an unseen predator, or hostile and unpredictable environment, in biological systems. We tried two different configurations with send-non-id, one that included max-known and send-id, and another that included max-known and send-self. Initial experiments that used the send-non-id penalty performed similarly to those described earlier. However, when we also changed the replacement strategy from mass-action to neighborhood, performance improved dramatically. The reason for this improvement is likely related to *kin selection* [93], which occurs when parent and offspring work together on cooperative tasks. In this case, parent and offspring are genetically similar, and thus likely to cooperate on the rewarded tasks, while avoiding the send-non-id penalty. We note that neighborhood replacement alone, without using either a penalty or cost, did not achieve the desired behavior.

Figure 3.3(a) shows the average messaging behavior using the tasks max-known, send-id, and the penalty send-non-id. In addition to the values plotted previously, we also plot MaxID, the number of messages sent that carry the largest ID in the population. Figure 3.3(b) is a detail of a single trial that shows improvement in the types of messages present in the population. Here we see that the number of junk messages has been dramatically reduced, and that the number of messages containing IDs greater than that of the sender is increasing, although slowly. Still, very few messages contain the largest ID.

Figure 3.4(a) shows average messaging behavior using the tasks max-known, send-self, and the penalty send-non-id. For the first time, we see evidence of the convergence of message types, where the number of messages carrying an ID greater than the sender's approaches the total number of messages sent. We observe that due to the send-self task, each organism



(a)



(b)

Figure 3.3: Average messaging behavior with the addition of the send-non-id penalty (a), and detailed messaging behavior of a single trial (b).

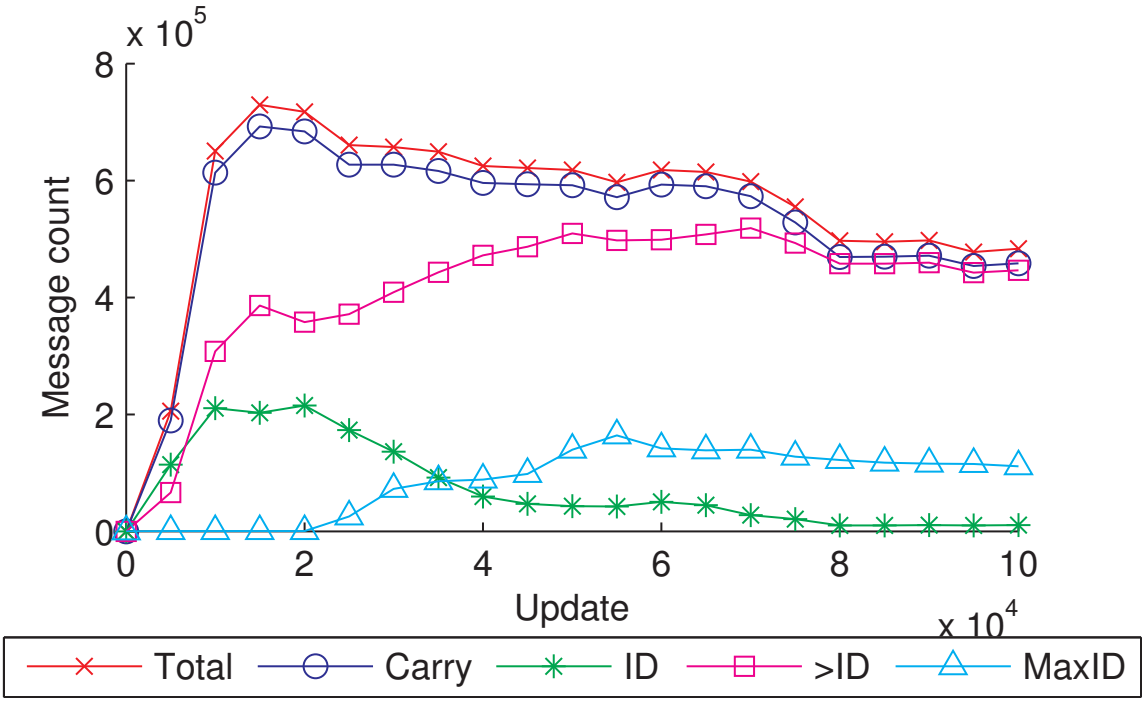
sends its own ID at least once, and we also see a significant number of messages that carry the largest ID. Figure 3.4(b) plots details of a particular trial where nearly all messages contain cell-IDs greater than that of the sender.

Moreover, those containing the largest ID (MaxID) represent 98.3% of all sent messages. It is this behavior, where nearly all sent messages converge to the largest ID, that was sought. We note that the drop in total number of sent messages corresponds to the evolution of filtering; we analyze a genome that exhibits such filtering behavior shortly.

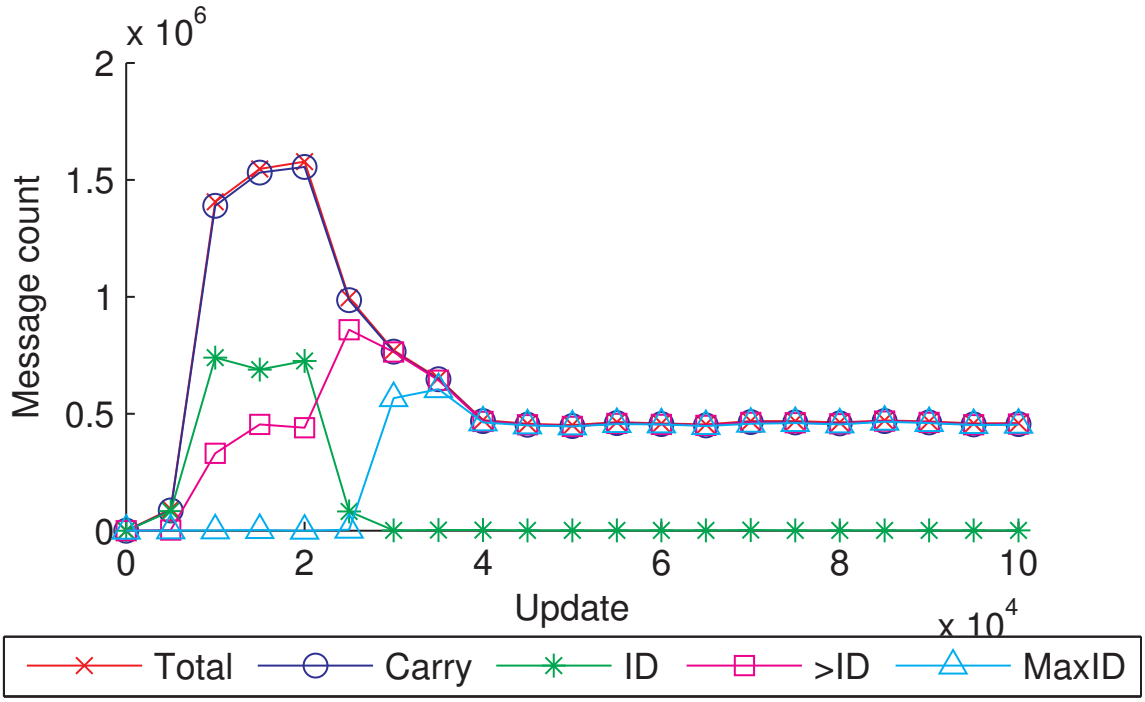
3.2.3 Recovery from ID Reset

Having determined that populations of digital organisms could cooperate to discover the largest ID, we next investigated whether the population could react to a change in that ID. Using the penalty-based task configuration described earlier, we added an event, reset-id, that when executed, resets the largest ID in the population to a smaller random value. Figure 3.5(a) shows the resulting average messaging behavior with the reset-id event configured to occur at update $50,000$, using tasks max-known, send-self, and send-non-id. Figure 3.5(b) shows the details of a single trial that recovered from changing the largest ID. In these figures, we see that the populations are not only able to recover from the change to the largest ID, but that they exceed their pre-reset levels within 10,000 updates.

Figure 3.6 shows keys stages of messaging behavior from Figure 3.5(b). Figure 3.6 comprises snapshots of the population during the evolution process. Each snapshot identifies which organisms are sending the largest ID, which are sending the second-largest ID, and which sent both during their lifetime. By frame (d) nearly all organisms are sending messages carrying the largest ID; a few organisms near the cell with the second-largest ID are

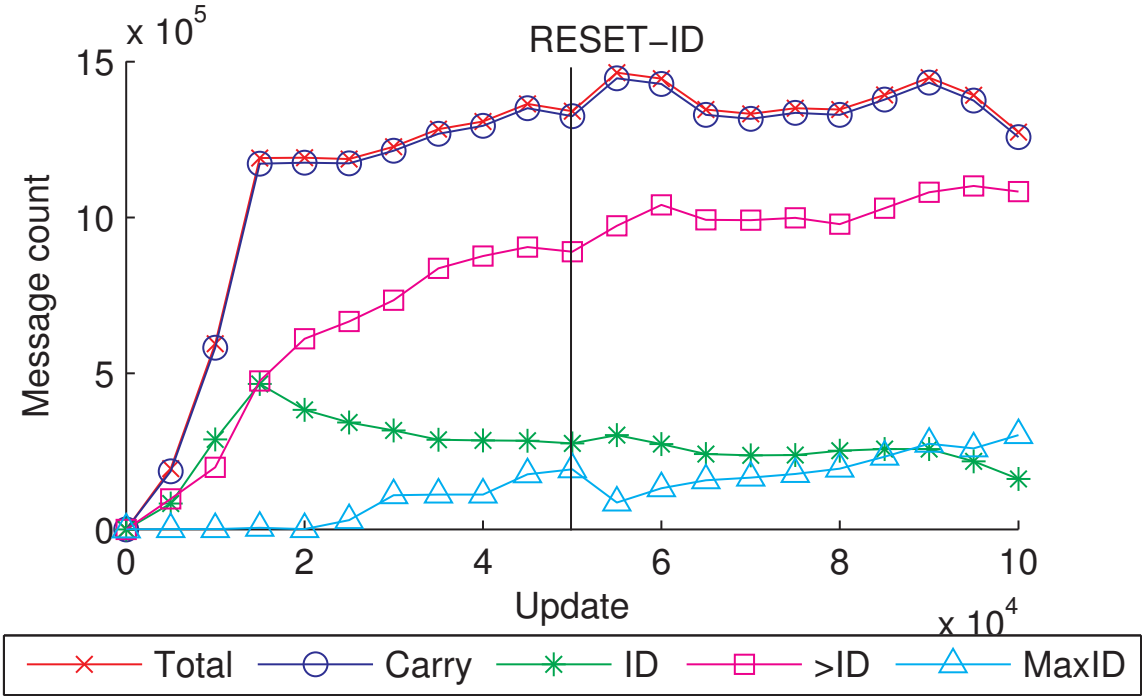


(a)

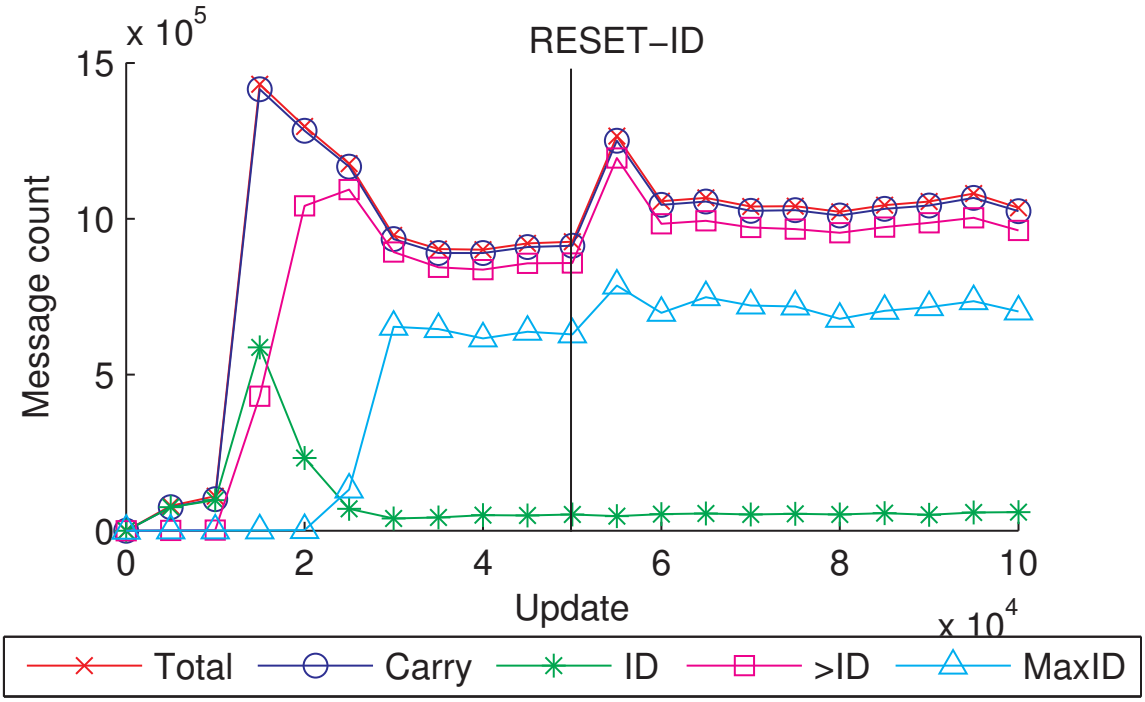


(b)

Figure 3.4: Average messaging behavior with a penalty using send-self (a), and detailed messaging behavior of a single trial (b).



(a) Average messaging behavior with a resets occurring at update 50,000.



(b) Messaging behavior of a representative trial.

Figure 3.5: Recovering from a change to the largest ID.

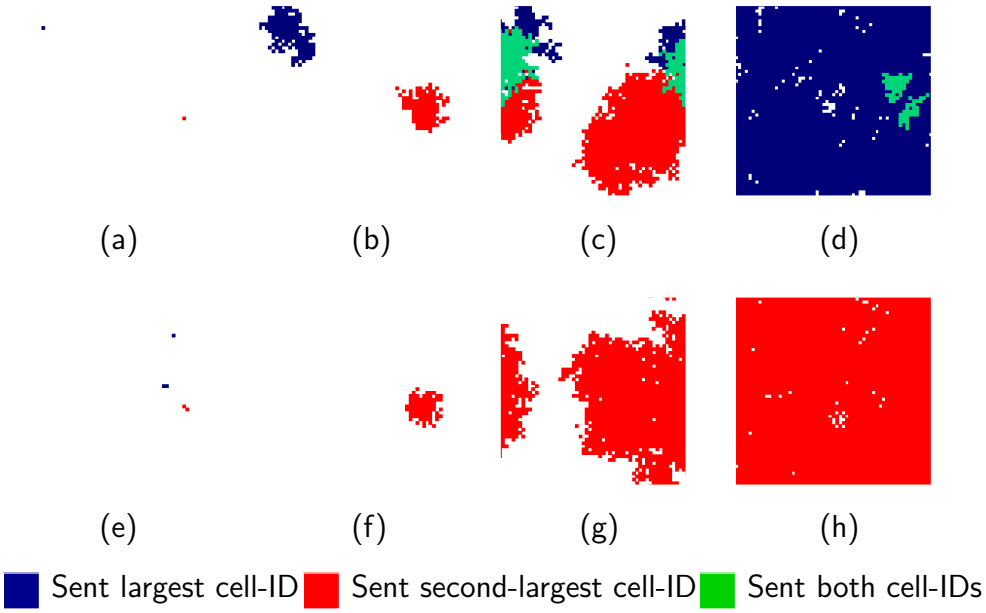


Figure 3.6: Eight frames excerpted from an AVIDA trace, demonstrating the evolution of distributed problem solving. Colors represent the messaging activity of organisms within each cell.

sending both. The largest ID is reset just prior to frame (e). As shown, the transmission of the (old) largest ID dies out quickly. The population, however, is able to recover and quickly proliferate messages that carry the new largest ID.

3.2.4 Genome Analysis

Figure 3.7 shows the relevant portion of the dominant genome responsible for the behavior shown in Figures 3.5 and 3.6. In this figure we have identified neutral mutations (shown shaded gray) as well as those parts of the genome that are relevant to determining the largest ID and the replication cycle. This particular genome comprises 84 instructions, of which 12 are responsible for the desired behavior, 22 are responsible for the organism's replication cycle, 1 instruction is shared, and 51 instructions, or 61% of the genome, are neutral mutations. An interesting feature of this particular genome is that its replication is

dependent upon it receiving a message that carries a ID larger than its own. In other words, organisms with this genome have evolved to the point where they depend upon the behavior of other organisms for their very survival. Specifically, if these organisms do not receive a message that has a data field larger than their own ID, *they will not reproduce*.

It should be noted that one of the forces at work in evolving this behavior is the selection of organisms that do *not* perform the send-non-id task. As soon as the reset-id event is triggered, any organism that sends the original largest ID is subject to the penalty for sending a junk message. Even in the absence of an explicit penalty, organisms that send the original largest ID would still not receive the reward for the max-known task. It is these *selective pressures* that are primarily responsible for the distribution of the largest ID. Moreover, an organism cannot be rewarded for sending a message containing the new largest ID without first having been sent that ID in a message (unless, of course, the organism lives in that cell). In other words, to survive a change in the largest ID, the organisms depend on cooperation.

The success of this study depended on carefully designed environmental conditions, specifically the presence of tasks that were able to reward *individual* organisms relative to the *global* state of the population. For example, the send-non-id task penalizes individuals for sending junk messages, but doing so requires knowledge of all IDs present in the population at any point in time. In the next section, we use group selection to eliminate the need for tasks that require perfect information of the global state.

3.3 Leader Election via Group Selection

In this section we again focus on leader election, but we use multilevel selection to decouple the desired cooperative behavior from the specific actions that must be performed by

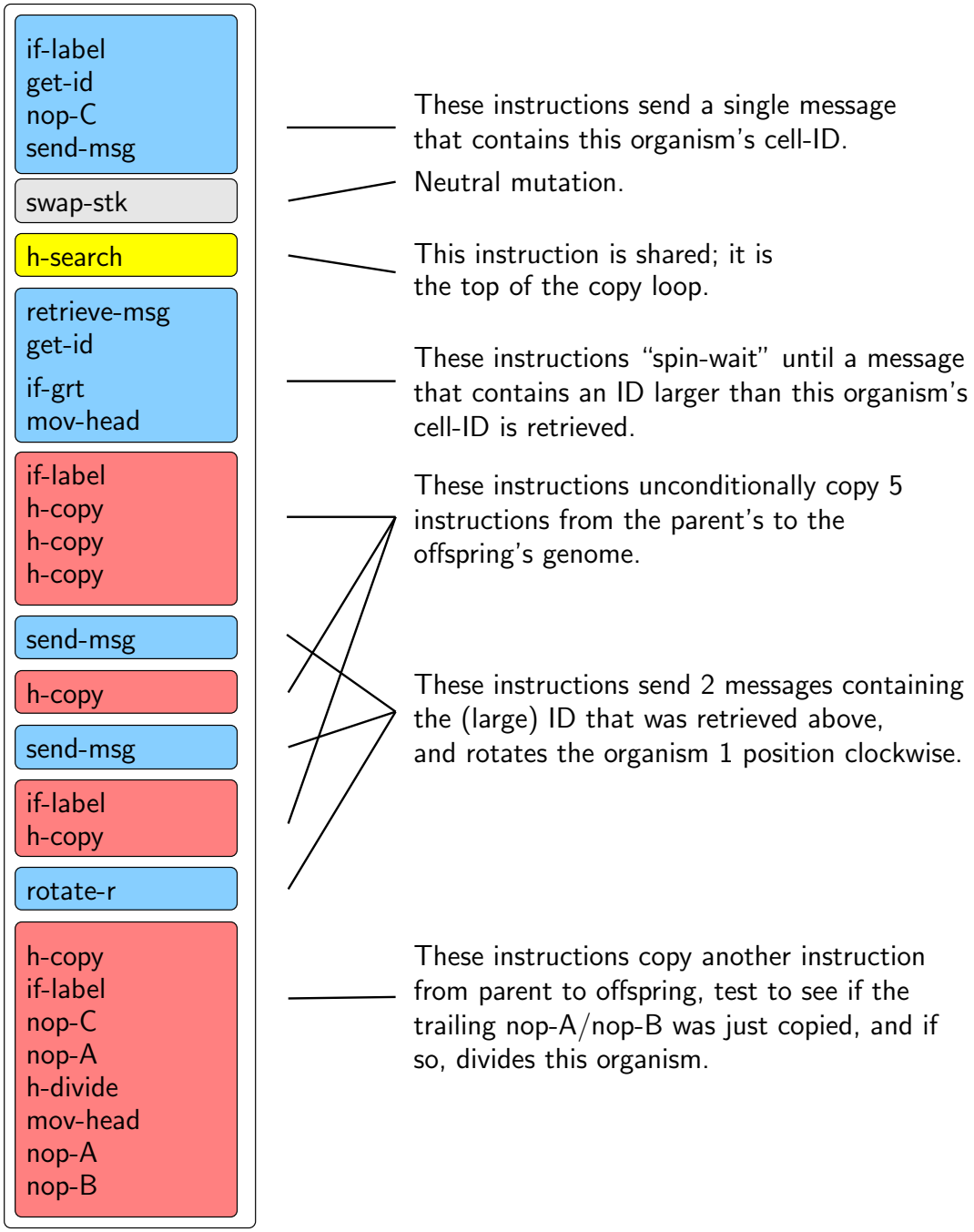


Figure 3.7: Portion of dominant genome sending messages that carry the largest ID.

individual organisms within the population. The strength of this approach is that, by not explicitly specifying constituent behaviors, we enable evolution to discover novel strategies that might not otherwise have been apparent.

Methods. In Section 3.2, we configured Avida to use a single population of 3,600 digital organisms in a 60×60 torus. For this study, we configured Avida with a population of 100 demes, each comprising 25 digital organisms. Each deme is configured in a 5×5 torus, except where noted. Experiments are again run for 100,000 updates, 20 separate Avida trials were performed for each experiment, and mutation rates are unchanged (default Avida configuration). Similar to the previous study, we again make use of cell data as identifiers of individual organisms, and for brevity refer to them as IDs.

The experiments presented in this section made use of the CompeteDemes framework described in Section 2.3.3, however demes are allowed to be heterogeneous and individuals within demes self-replicate and are subject to mutation (that is, digital germlines are not used). For each of the following experiments we have provided a fitness function that takes the behavior of a deme as input, and outputs a floating-point number (the deme's fitness) that is then used to determine which demes are allowed to replicate. Here, each deme is replicated via fitness-proportional selection, and all organisms within a replicating deme are copied to the replaced deme.

3.3.1 Multilevel Selection With Tasks

Our first experiment investigated whether leader election would evolve when deme competition was used in conjunction with individual tasks, specifically the max-id, send-self, and send-non-id tasks described in Section 3.2. We also defined a deme fitness function called

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

Figure 3.8 depicts maximum deme fitness averaged over 20 runs. We see here that the average deme was unable to elect an initial leader, since the fitness does not reach 625, although up to 20 out of 25 organisms did agree on a leader.

Figure 3.9 shows the different message types sent, again averaged over 20 runs. Here we see that the number of >Sender ID messages are roughly 50% of all messages sent, indicating that organisms are indeed forwarding IDs larger than their own, and thus performing leader election.

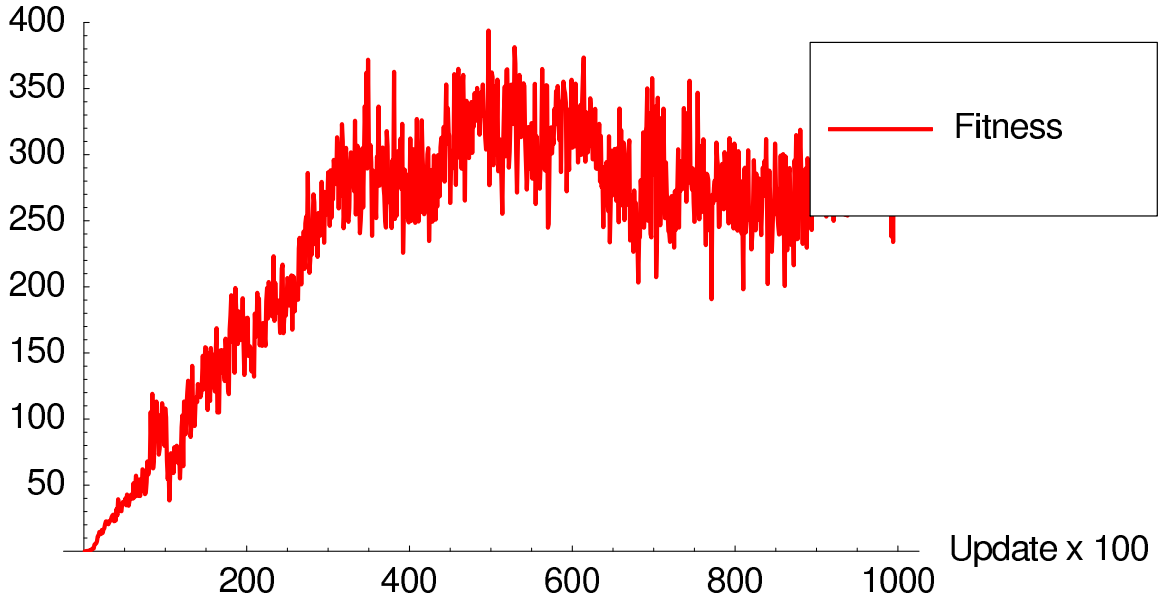


Figure 3.8: Maximum deme fitness using tasks and deme competition.

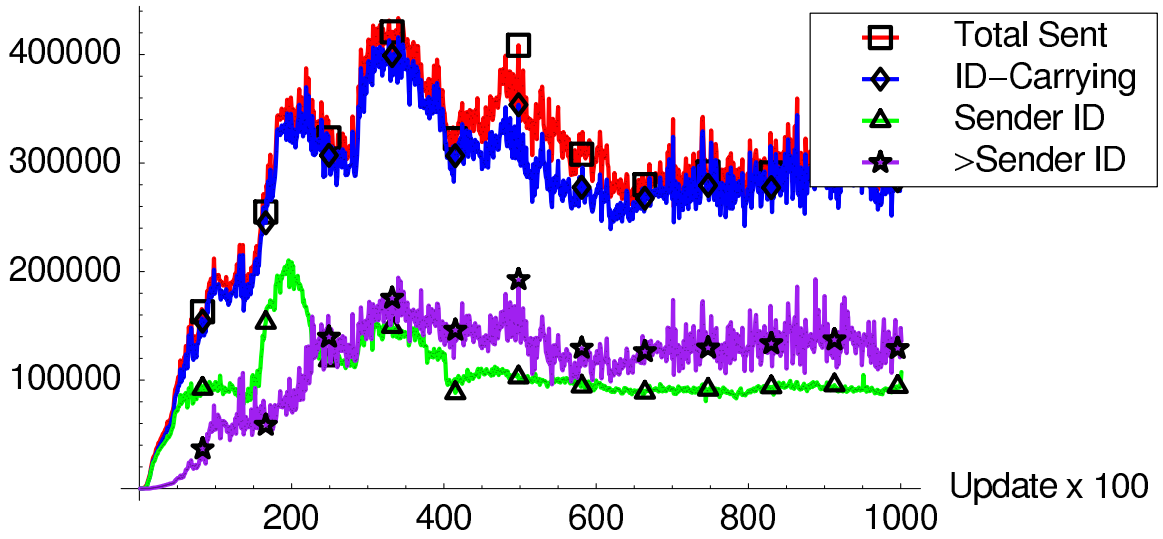


Figure 3.9: Messaging behavior using tasks and deme competition.

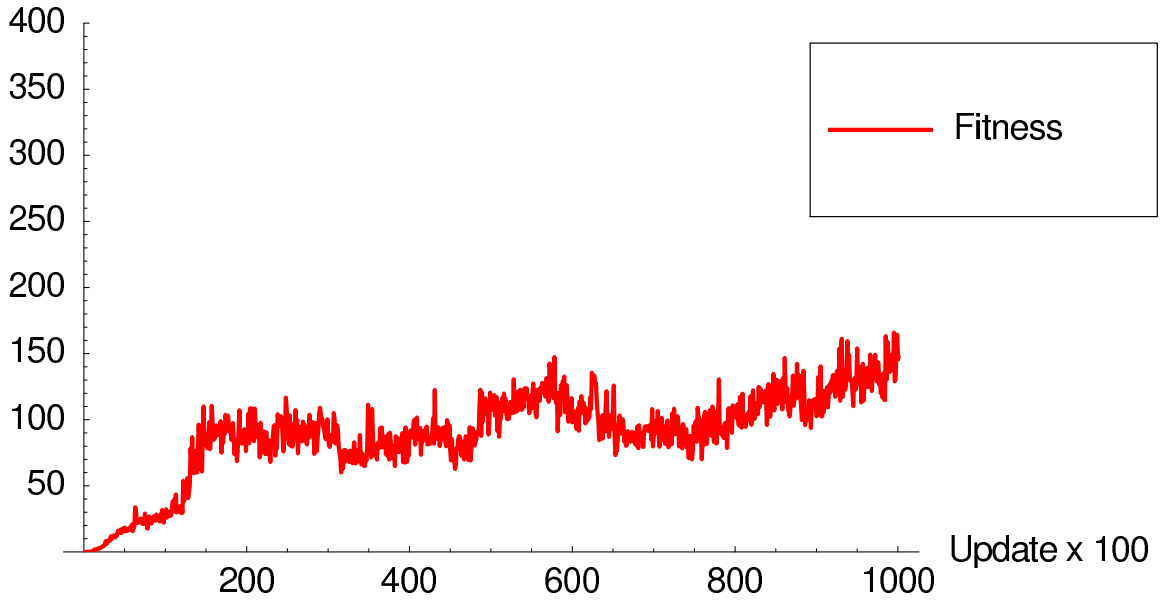


Figure 3.10: Maximum deme fitness without using tasks, with deme competition.

3.3.2 Multilevel Selection Without Tasks

Our next experiment removed the individual rewards for organisms that performed the max-id, send-self, and send-non-id tasks. This experiment tests to see if multilevel selection alone, without rewarding for constituent behaviors, is sufficient to evolve leader election. Figure 3.10 plots the maximum deme fitness averaged over 20 Avida runs without rewarding for tasks. Here we see that fitness is generally increasing, although it does not indicate that leader election has occurred.

Figure 3.11 shows the message types averaged over 20 Avida runs. A notable difference between this experiment and those that include tasks is that 60% of messages do not carry an ID. However of those messages that carry an ID, 50% carry an ID larger than the sender's, indicating that the same leader election strategy as the previous experiment is being employed.

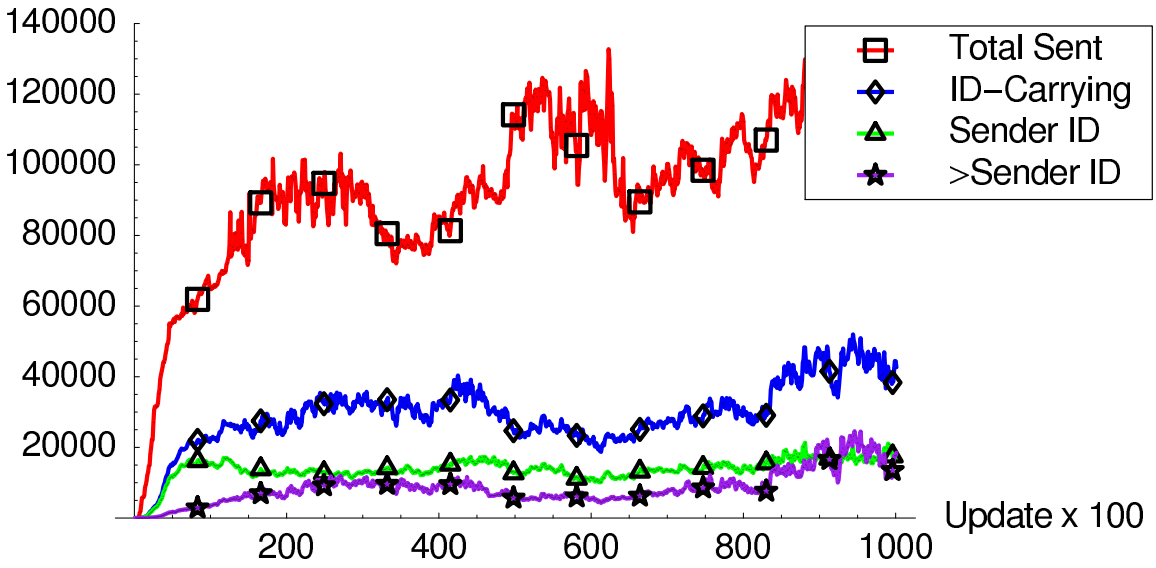


Figure 3.11: Messaging behavior without using tasks, with deme competition.

3.3.3 Multilevel Selection in a Clique

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

Figure 3.12 depicts the maximum deme fitness averaged over 20 Avida runs. Here we see that fitness is steadily increasing, and is an improvement over the average fitness shown in Figure 3.10. However, as in the previous experiment, no evidence of leader election is found.

Figure 3.13 shows the message types averaged over these same 20 Avida runs. Here we see that messages are now almost completely non-ID carrying, indicating that other means (e.g., the get-neighbor-id instruction) are being used to retrieve IDs. This experiment

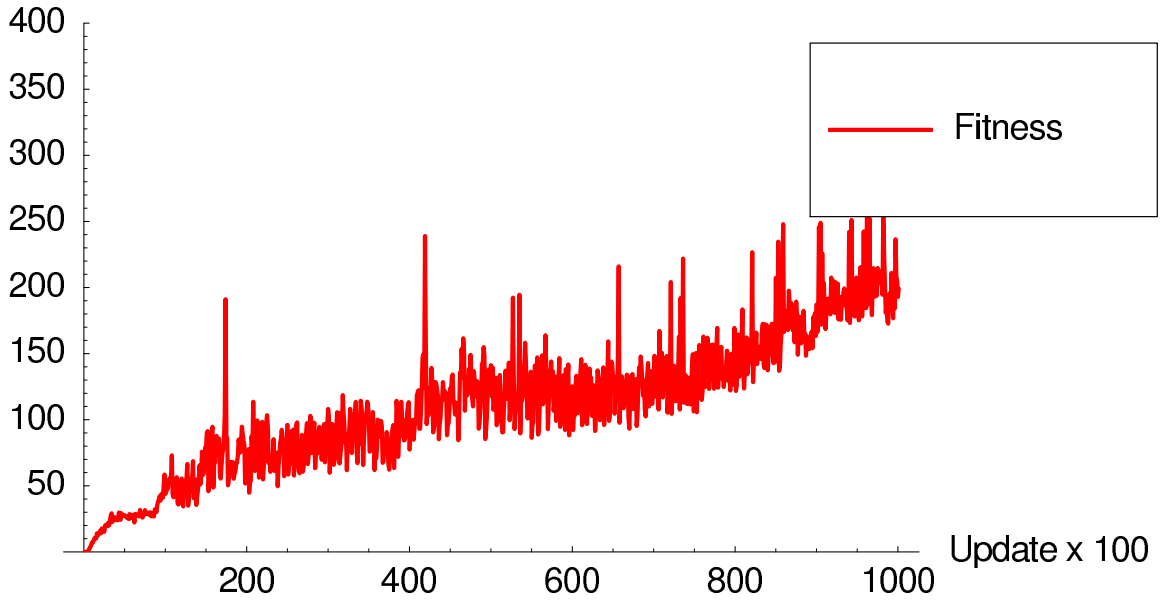


Figure 3.12: Maximum deme fitness without using tasks, with deme competition, in a clique topology.

indicates that both topology and the specific capabilities of the organisms are important factors for the evolution of leader election.

3.3.4 Leader Election With Neighbor Scanning

Using the rotate family of instructions, organisms in AVIDA have the capability to rotate through their list of neighbors, where the particular neighbors of an organism are dependent upon the topology. Variations of the rotate instruction include scan-rotate-{l,r}, where the organism will rotate to the first neighbor that contains a specified *label*. A label in AVIDA is a series of one or more nop-{a,b,c} instructions anywhere in the organism's genome. When a scan-rotate-{l,r} instruction is followed by a label, it will rotate left (counter-clockwise) or right (clockwise) until it faces a neighbor whose genome contains the *complement* label. Complements are formed from the following replacements: nop-a→nop-b; nop-b→nop-c; nop-c→nop-a. In every case, rotation stops at the first neighbor containing the complement, or

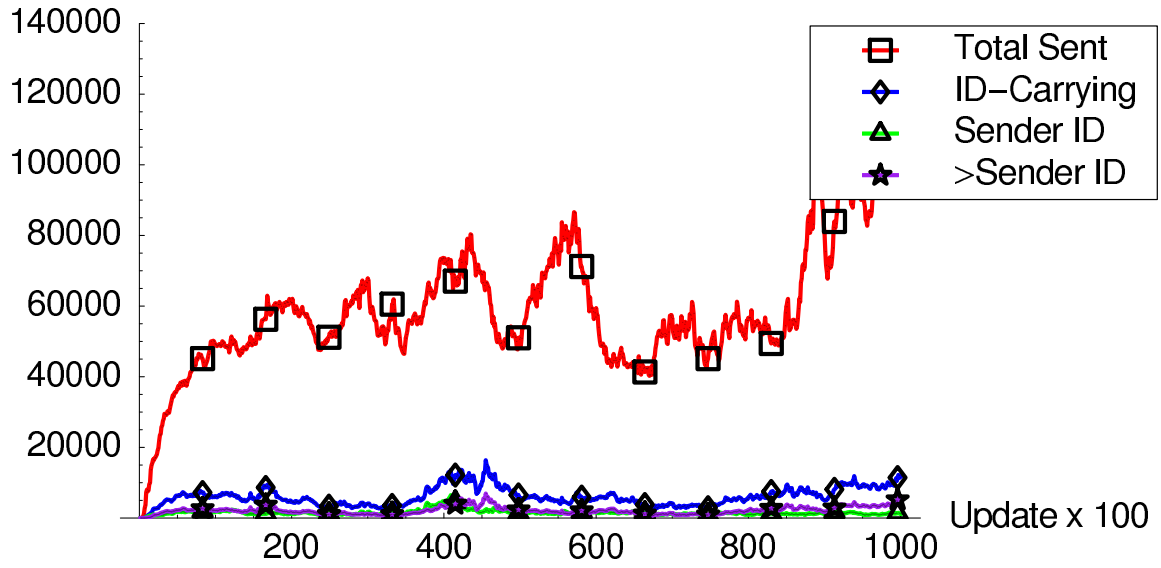


Figure 3.13: Messaging behavior without using tasks, with deme competition, in a clique topology.

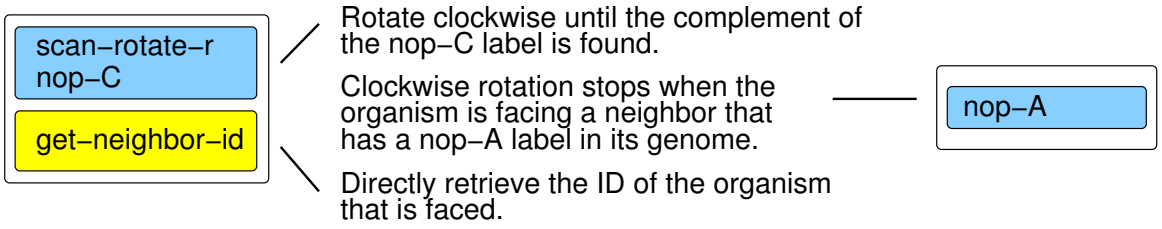


Figure 3.14: An example of the usage of labels in Avida.

at the original facing, whichever comes first. An example of the scan-rotate-r instruction and the use of labels is shown in Figure 3.14. This figure contains fragments of two genomes, where the three instructions in the left-most genome scan neighboring organisms for the label shown in the right-most genome. Once found, the get-neighbor-id instruction will place the ID of the faced neighbor directly into a register.

In this experiment, we configured Avida to use the scan-rotate-{l,r} instructions, while keeping the topology a clique. Figure 3.15 depicts the maximum deme fitness averaged over 20 Avida runs (please note the scale change). Here we see a significant improvement over previous experiments, with the average deme electing just under 3 leaders in 100 updates (3

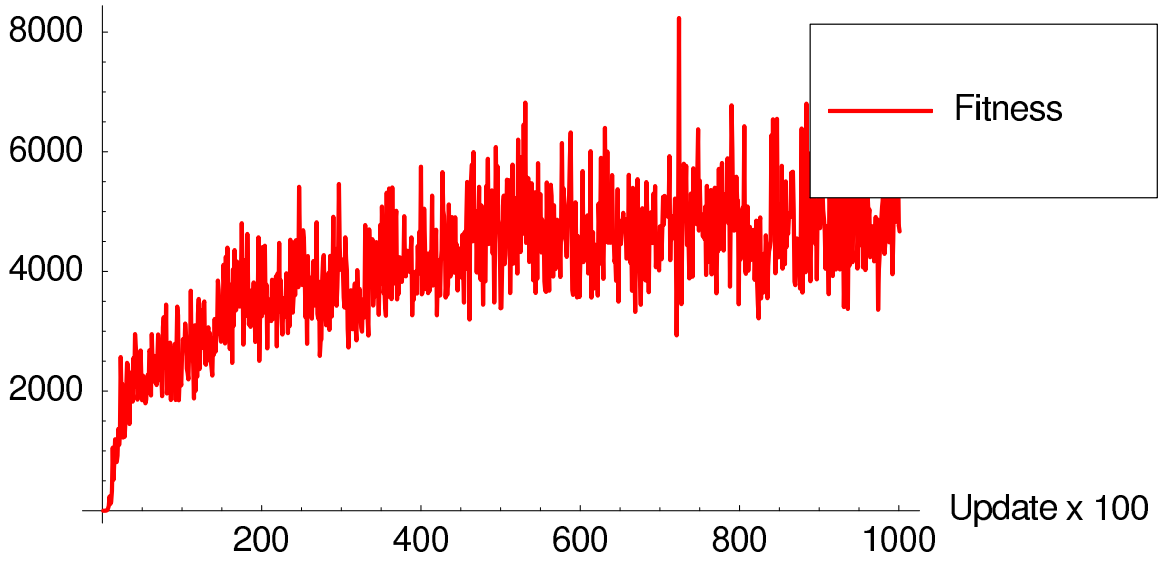


Figure 3.15: Maximum deme fitness without using tasks, with deme competition, in a clique topology, with neighbor scanning.

rounds of leader election is represented by a fitness of 5,625).

Figure 3.16 depicts the messaging behavior averaged over these same 20 AVIDA runs, where we see that again, the majority of messages do not carry an ID. So, while organisms are successfully electing leaders, they are apparently (and surprisingly) not using messages to do so!

3.3.5 Genome Analysis

We investigated this result by examining the best-performing deme out of all AVIDA runs, which elected 9 leaders during a 100 update period. Figure 3.17 shows the fitness of that deme, which reaches a maximum fitness of 51,076 at update 78,900.

Figure 3.18 shows the 12 different genomes present in this deme during the election of a leader (13 of the 25 organisms in this deme shared a genome). Each genome contains a common sequence of instructions that implements leader election, shown at the top of this figure. Specifically, each organism rotates clockwise, stopping at the first organism that

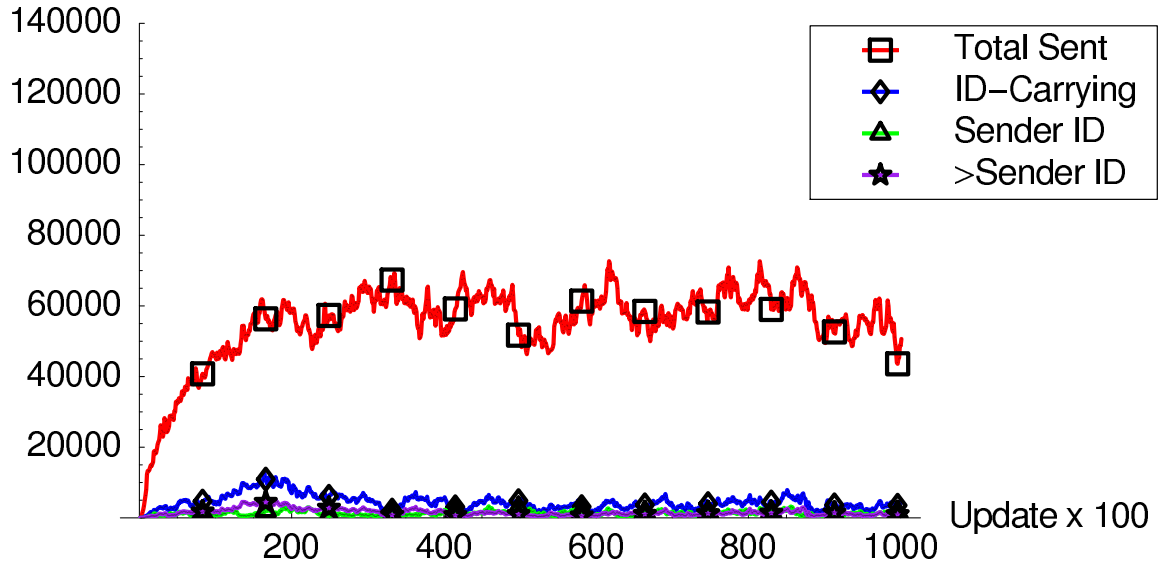


Figure 3.16: Messaging behavior without using tasks, with deme competition, in a clique topology, with neighbor scanning.

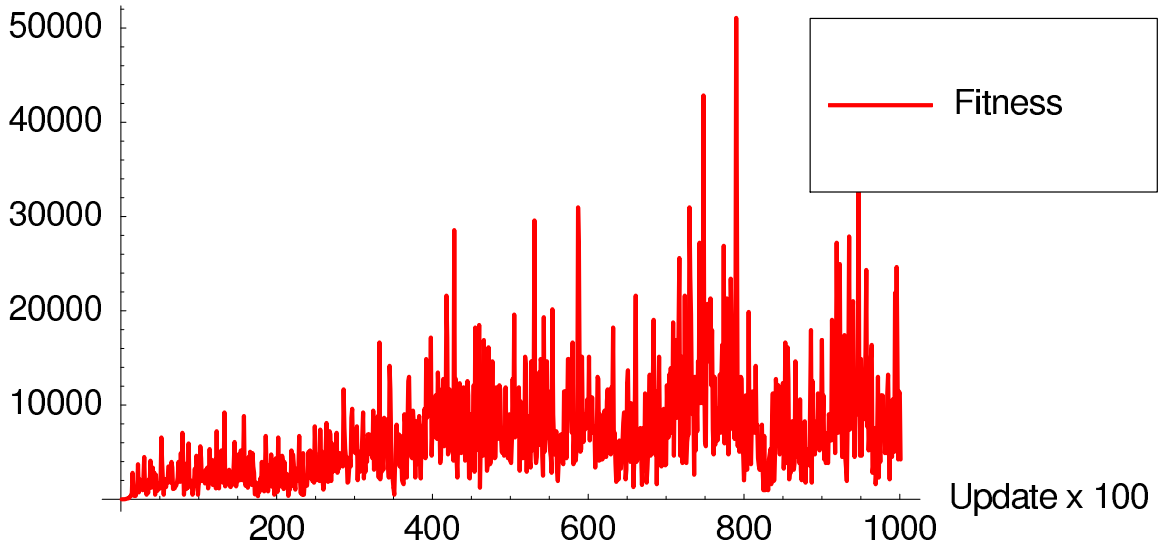


Figure 3.17: Maximum fitness of the best-performing population using deme competition, in a clique topology, using neighbor scanning. Maximum fitness of 51,076 occurs at update 78,900.

contains a nop-A label (the complement of the nop-C label). It then sets its leader ID to the ID of the faced organism. Every time this deme elected a leader, there was exactly one organism in the deme that contained a nop-A label, shown at the bottom of this figure. In other words, a leader was elected whenever exactly one organism contained a specific identifying feature that the group had agreed upon. Interestingly, this approach is similar to the immunological question of detecting "non-self," where an immune system will attack cells that are identified as not belonging to the host [115].

As a result of these studies, we can conclude that digital evolution, in combination with multilevel selection, is capable of evolving an algorithm for leader election. However, we observe that the result of this study - a group of genomes that depends on mutation - is not one that we can reasonably expect to deploy in a real-world system. For example, one is unlikely to deploy a mobile *ad hoc* network that is capable of mutating its behavior! However, it does illustrate that evolution can and will find solutions beyond the preconceptions of the developer! For this reason, we now turn to studies of a general form of consensus that does not require on-line mutation.

3.4 Simple Consensus

Our next study of data-driven behavior focused on a general form of consensus that we call the *Simple Consensus Dilemma* (SCD). Here, in contrast to previous studies of leader election, we framed the consensus problem in such a way that the results should be implementable, that is, the output of the evolutionary process should be an algorithm that we could expect to deploy in a real-world computing system. The main difference between this study and those previous is that we no longer allow individual organisms to self-replicate or

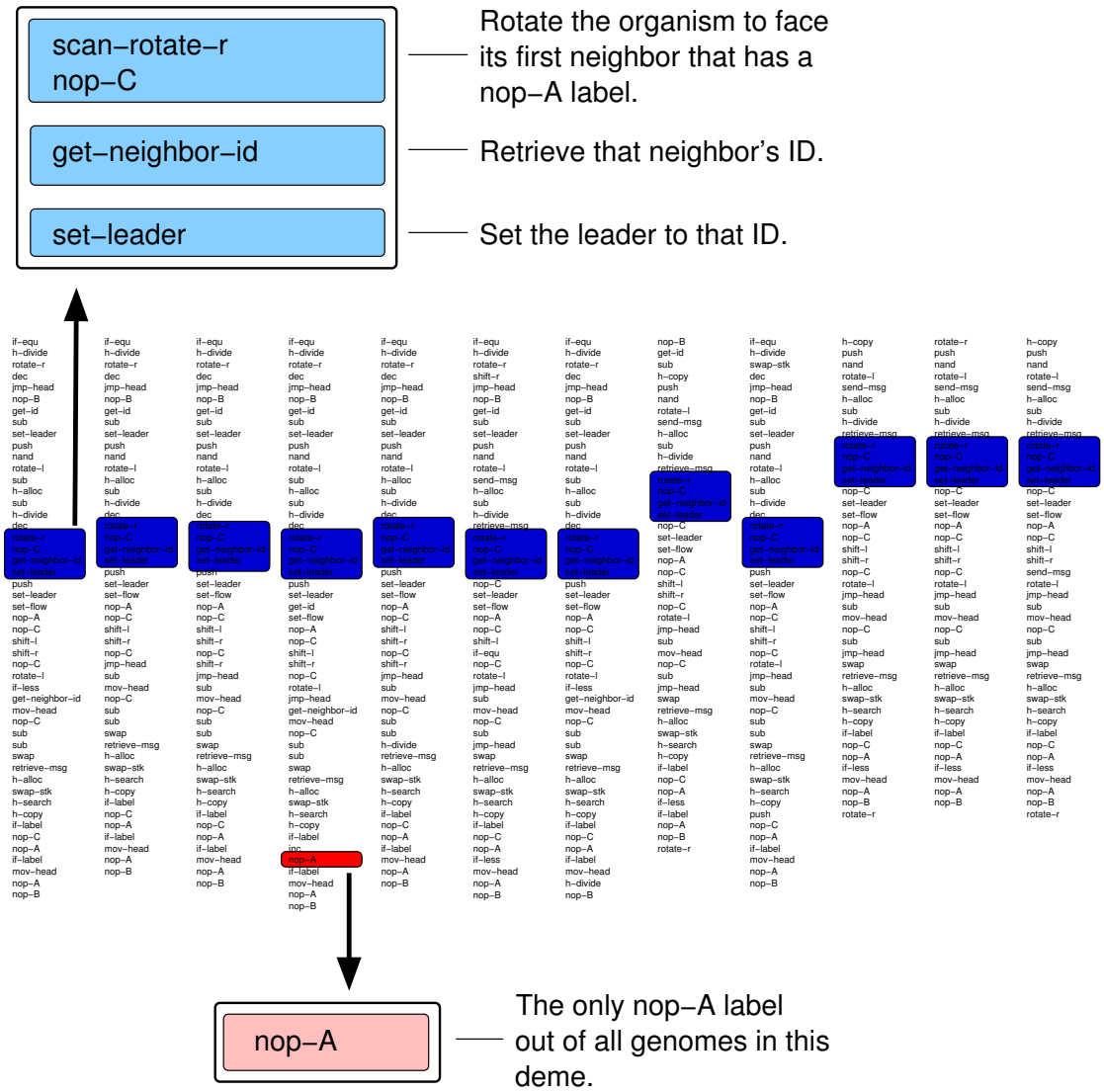


Figure 3.18: Depiction of the 12 different genomes active in the deme that elected 9 leaders. 13 organisms shared a genome, while exactly one organism had a genome containing the nop-A instruction, which indicated the leader.

mutate. Instead, we used deme competition and digital germlines (described in Section 2.3.3) to force homogeneity within each deme. This approach guaranteed that the final dominant genome encoded a solution that we could deploy on multiple systems, with a global behavior emerging from their individual actions.

Methods. The SCD is based on the n-process id consensus problem [46, 116], itself a variation of the alignment problem [117,118], where a group of agents seek to reach agreement upon a common agent id. Informally, we define the SCD as follows[1]:

> Each agent in the population is assigned a unique identifier. Agents are independent, and can communicate by sending messages. Each agent can also designate a value, selected from the set of all identifiers, as its "opinion." After some period of time, the opinions of all agents are examined. Agents that express the same opinion are rewarded, while agents that express different opinions are punished. How should the agents act?

Necessarily, solving the SCD requires cooperation and communication among agents, as well as a coordinated strategy for selecting a value (opinion) to agree upon. The question we are asking then, is: Can evolution solve the SCD, and if so, what strategies will be employed?

All new instructions employed in this study are summarized in Table 3.2; the communication and cell data instructions described in Section 2.3.2 were also used. Of particular note are the instructions associated with "opinions" and the "flash" capability. These instructions enable organisms to manipulate their opinion register, used as the basis for the fitness functions that will be presented later; and to both sense and trigger virtual flashes, a synchronization primitive based on the behavior of fireflies, respectively (flash will be described

[1]This 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 [58].

Table 3.2: Relevant instructions for this study. All instructions are equally likely to be selected as targets for mutation.

Instruction	Description
get-opinion	Sets register BX to the value of the caller's opinion register.
set-opinion	Sets the caller's opinion register to the value in register BX.
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, then execute the subsequent instruction. Otherwise, skip the subsequent instruction.
flash-info	If the caller has ever received a flash, then set BX to 1 and CX to the number of cycles since that flash was received. Otherwise, set BX and CX to 0.

in more detail in the following chapter).

We configured Avida to use the CompeteDemes process, with specific configuration values summarized in Table 3.3. We 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. 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. We note that in this study, individual organisms are no longer required to self-replicate, as demes are fully populated from their germline.

We set each cell data to a random 32-bit integer, which was used as the ID for the organism in that cell. We then defined a fitness function to reward demes whose constituent

Table 3.3: 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

organisms set their opinion to a common ID. The specific fitness function used here was:

$$F = (1 + S_{max})^2 \tag{3.1}$$

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 IDs present within the deme contribute to the deme's fitness.

Results. Figure 3.19 plots the average and best-case performance of individual demes across 30 trials. In this figure, the y-axis represents the fraction of agents agreeing ($S_{max}/25$), where a value of 1 is complete consensus, and the x-axis is in units of updates. In these trials, an update corresponds to each organism receiving, on average, five virtual CPU cycles. Here we see that the best-performing demes achieved consensus after approximately 25,000 updates, while the average deme steadily approached consensus.

Figure 3.20 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

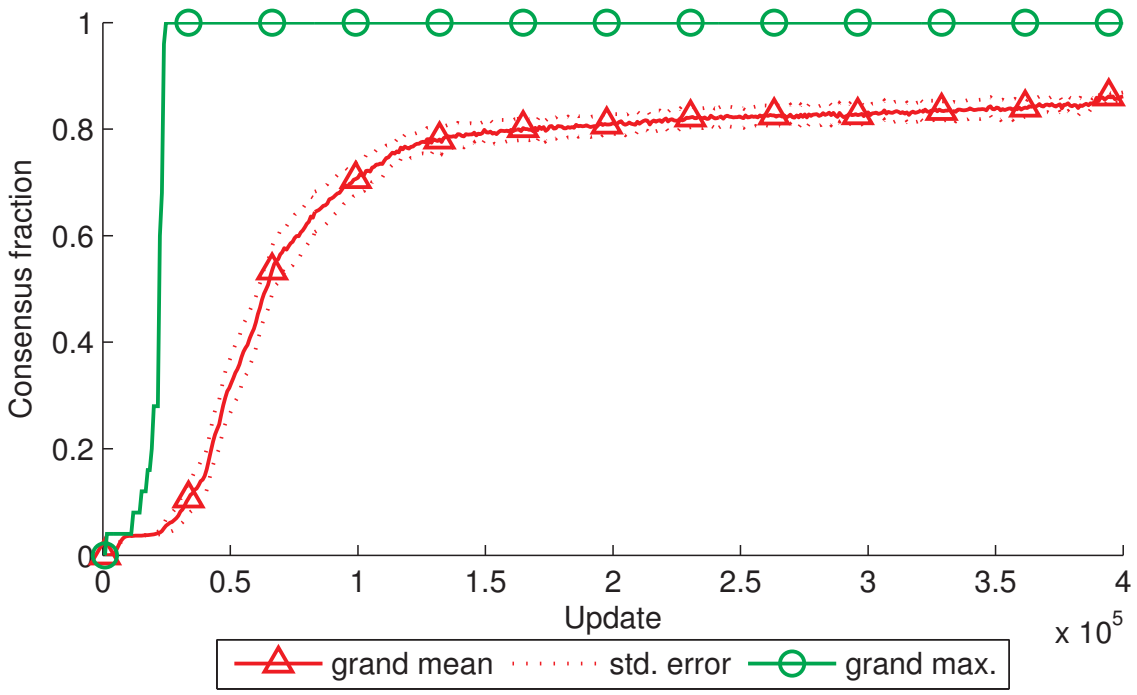


Figure 3.19: Average and best-case deme performance, in terms of consensus. Consensus is first achieved near update 25,000.

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 genomes appeared to employ a strategy that searched for the maximum ID, while an additional five genomes searched for the minimum ID. These approaches are in fact similar to the evolved strategy described in Section 3.2. Whereas in that study we used Avida tasks, however, here 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 3.21 depicts a fragment of an evolved genome that solved the SCD by searching for the maximum ID. At the individual level, this genome caused the organism to set its opinion

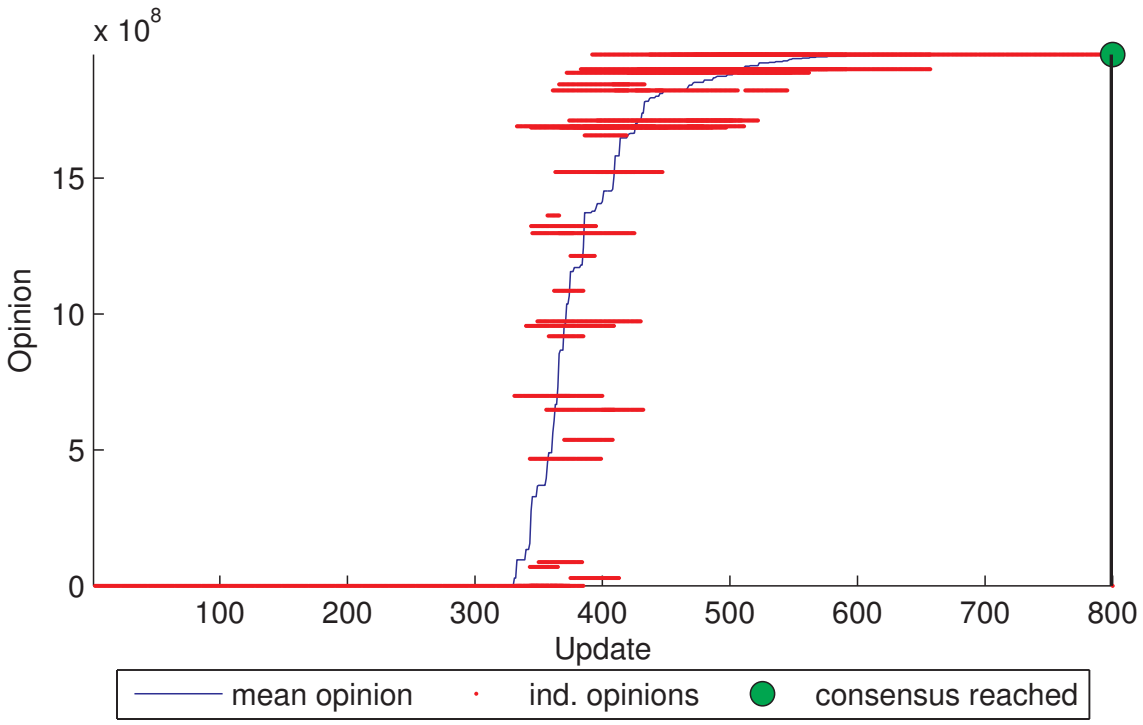


Figure 3.20: Representative behavior of a single deme while reaching consensus. This behavior was produced by testing the dominant genome from one of 30 trials.

to either its own ID, or the contents of a received message, whichever was larger. When a group of organisms all share this genome, each eventually set their opinion to the maximum ID of the group. Of course, while such an approach solves the SCD, it is *brittle*. Specifically, the system cannot recover from the removal of the maximal ID, nor can it detect when that ID is replaced. For this reason, we next studied a variation of the SCD that incorporated IDs that changed over time.

3.5 Iterated Consensus

Our next study of data-driven behavior focused on an iterated version of the Simple Consensus Dilemma, which we called the *Iterated Consensus Dilemma (ICD)*. As will be seen, not only do we verify that digital evolution is capable of discovering distributed behaviors,

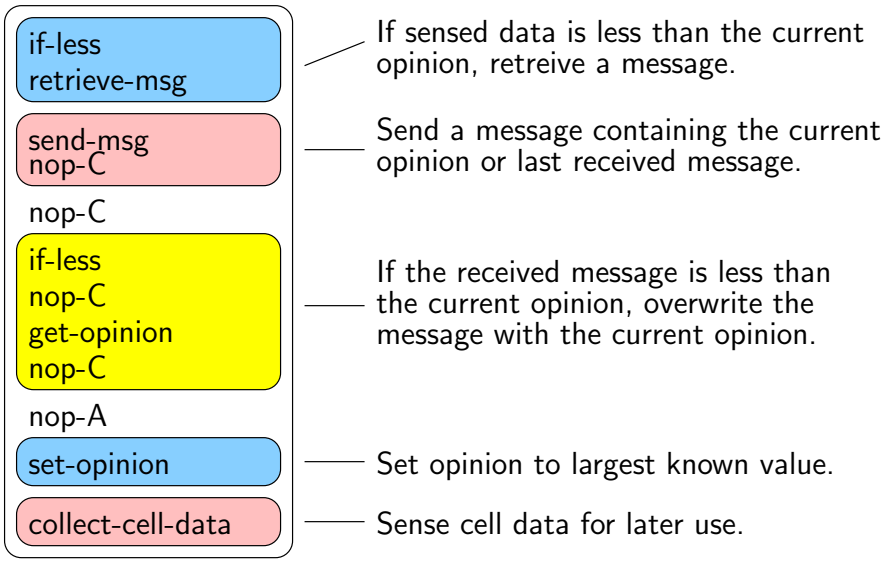


Figure 3.21: Genome fragment responsible for searching for the maximum ID present within a deme.

but we also show that AVIDA is capable of producing novel heuristics for consensus.

Methods. The ICD modifies the SCD by introducing *rounds*, similar to the Iterated Prisoner's Dilemma, 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 previously agreed upon is no longer valid, and then recover to reach consensus on a different value. The 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 3.4. The specific fitness function we used here was:

$$F = (1 + S_{max} + R \cdot D)^2 \tag{3.2}$$

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 the end of every update to determine if its constituents have reached consensus. If so, the value of R for that deme is incremented. We note that organisms can set their opinion multiple times during a single update; only the value of the opinion register at the end of the update is used to calculate R. Whenever consensus is reached, the ID corresponding to the agreed-upon value is reset to a random integer from the range bounded by the maximum and minimum ID present within that deme, and the organism in that cell is replaced (killed and overwritten) with a new organism instantiated from the germline. Thus, this fitness function rewards demes for reaching consensus as frequently as possible within a single deme competition period. In all other respects, the experiments described here are configured identically to those in Section 3.4.

3.5.1 Initial Solutions

Figure 3.22 plots the average and best-case performance of individual demes across all 30 trials. Here, the y-axis is units of consensus rounds, and the x-axis is again in units of updates. The data show that the average deme approached a single consensus round per competition period, while the best-case performance of any deme was two consensus rounds.

3.5.2 Improved Solutions

Based on 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

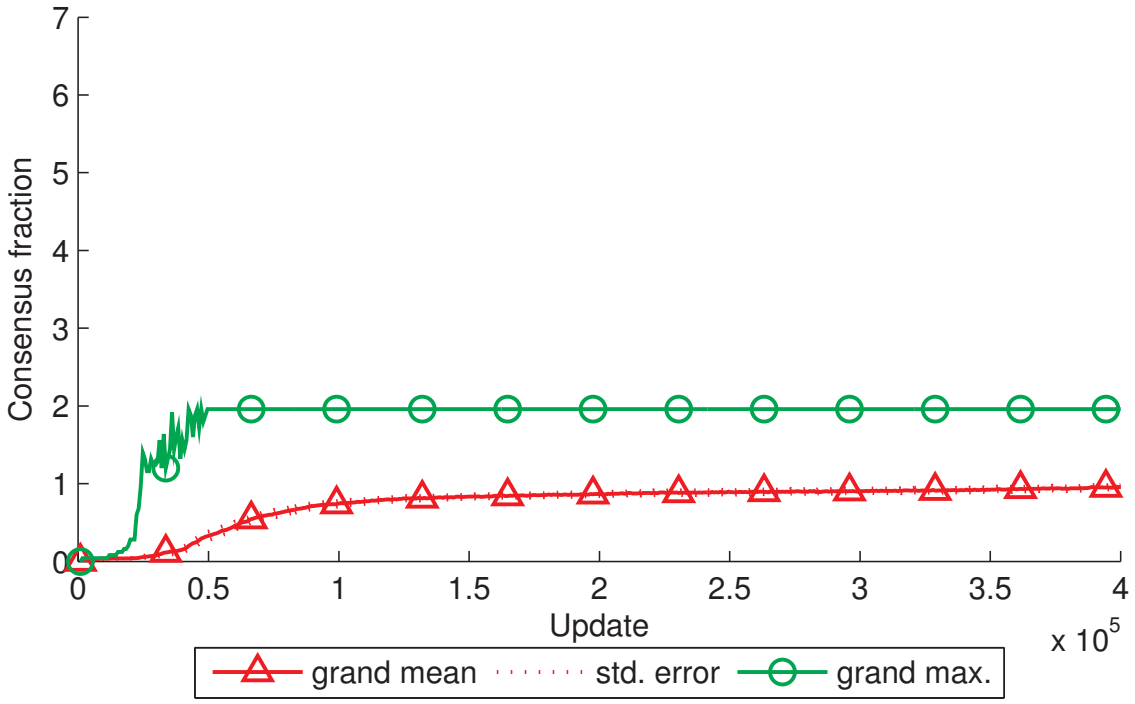


Figure 3.22: Average and best-case deme performance, in terms of consensus. The first deme to pass multiple consensus rounds occurs near update 50,000.

effect of a slight background rate of death on the evolution of consensus. 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 3.23 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 (Wilcoxon rank sum test, $p < 0.001$), the best-case performance of an individual deme oscillated between three and four complete consensus rounds, a marked improvement over the previous results. Based on these results, we concluded 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.

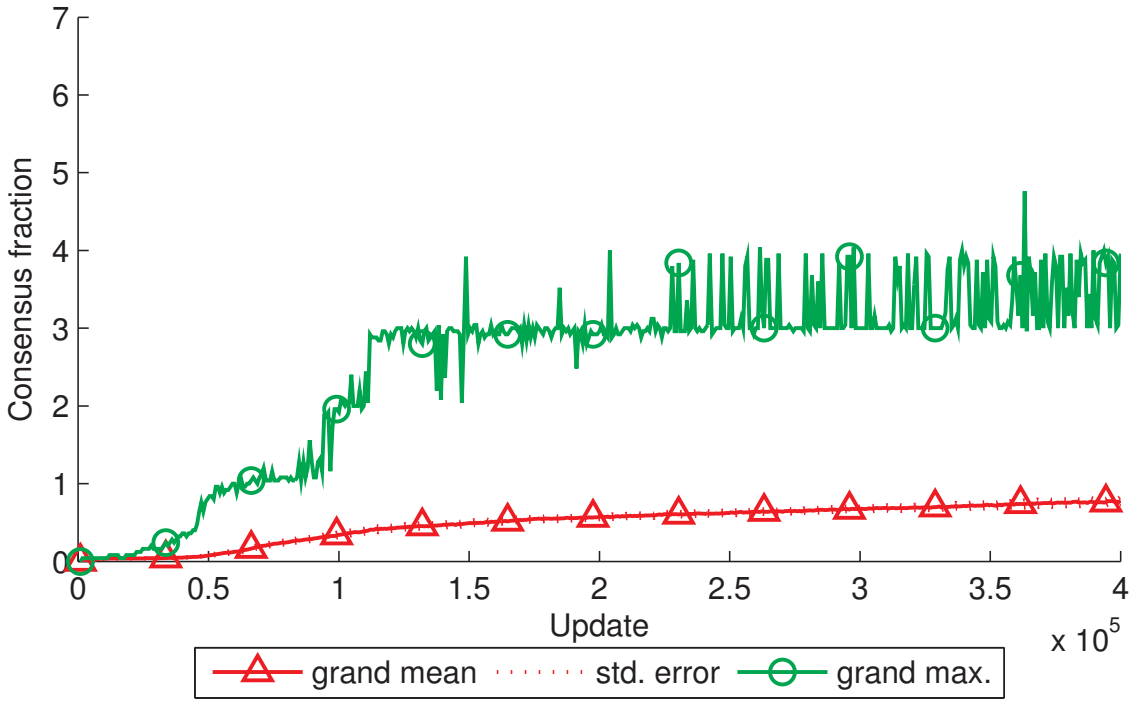


Figure 3.23: Average and best-case deme performance for the Iterated Consensus Dilemma, with the addition of death during deme competition periods.

We then hypothesized that the evolution of a solution to the ICD might not simply depend on data-driven behavior, but that an evolved solution might also include some element of temporal behavior. Thus, for our next experiment, we added synchronization primitives to the instruction set. Specifically, we added the flash and flash-info instructions, which broadcast virtual "flashes" (similar to fireflies in biology) to an organism's neighbors, and retrieve information about any sensed flashes, respectively. We also provided organisms with the bcast1 instruction, enabling them to broadcast messages to all of their neighbors with a single instruction. We note that addition of bcast1 alone did not significantly improve consensus, nor did the inclusion of instructions related to sensing the state of an organism's neighbors.

Figure 3.24(a) plots the average and best-case performance of individual demes across

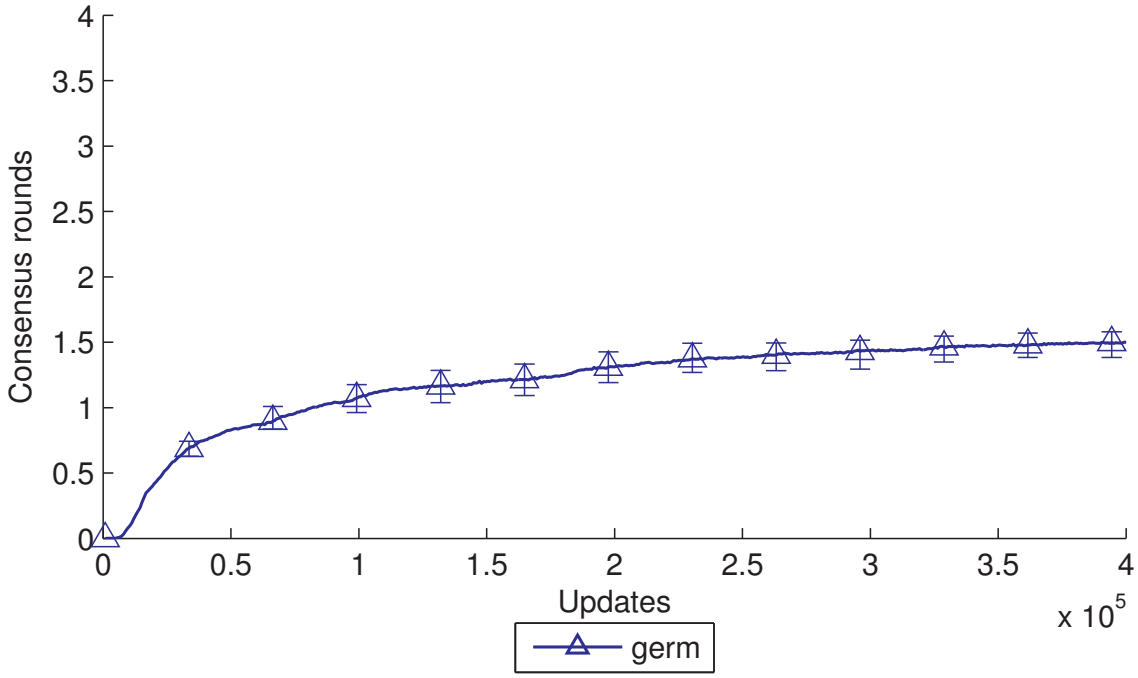
all 30 trials, and Figure 3.24(b) depicts the specific behavior of a single deme, including the individual opinions (red points), mean group opinion (solid blue line), when consensus is reached (solid vertical line), and the value of that consensus (filled circle). Here we see a dramatic improvement in both the average and best-case deme performance, where the average deme approached 1.5 consensus rounds, and the best-case performance of any deme achieved 5 consensus rounds. An interesting result brought to light in this experiment was that the combination of broadcast, sensing, and synchronization instructions were needed as building blocks for the behavior shown in Figure 3.24(b), although the genome responsible, described in the next section, does not include the flash instruction.

3.5.3 Genome Analysis

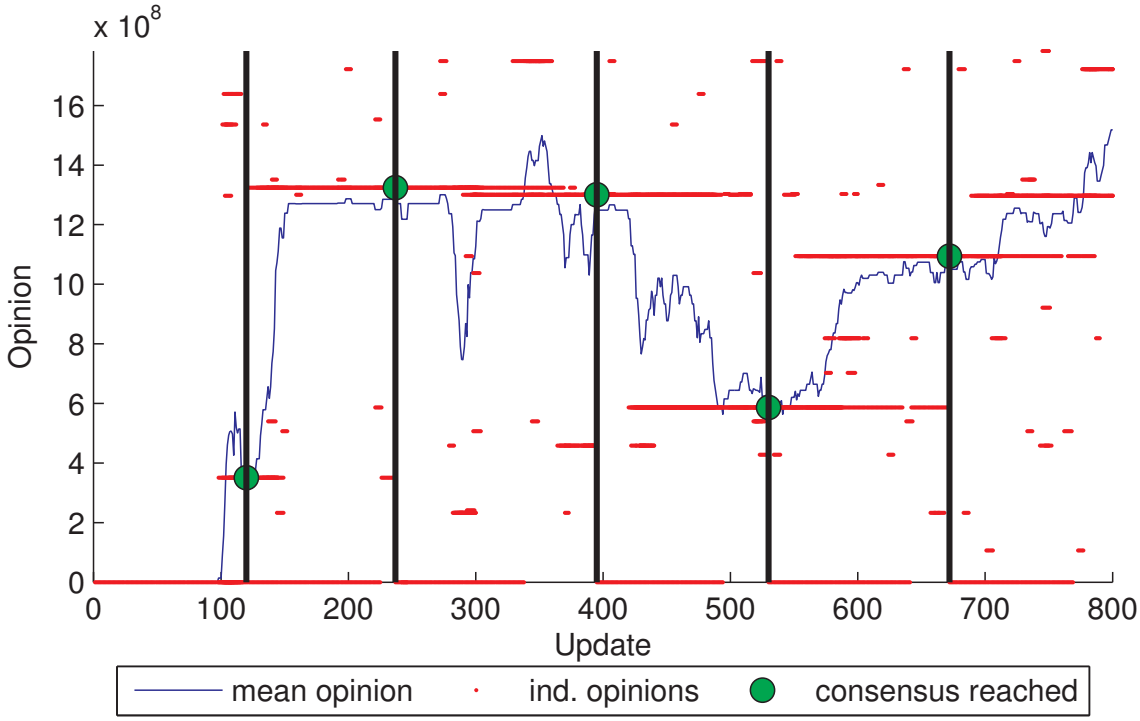
Figure 3.25 shows the annotated genome responsible for the behavior in Figure 3.24(b). Instructions that do not affect the overall fitness of this genome are not labeled. We observe that this genome evolved to use a fairly short (36 out of 86 instructions) loop to solve the ICD.

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.

Algorithm 3.1 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



(a) Average and best-case deme performance.



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

Figure 3.24: Deme performance and detailed behavior for the Iterated Consensus Dilemma, with the inclusion of synchronization instructions.

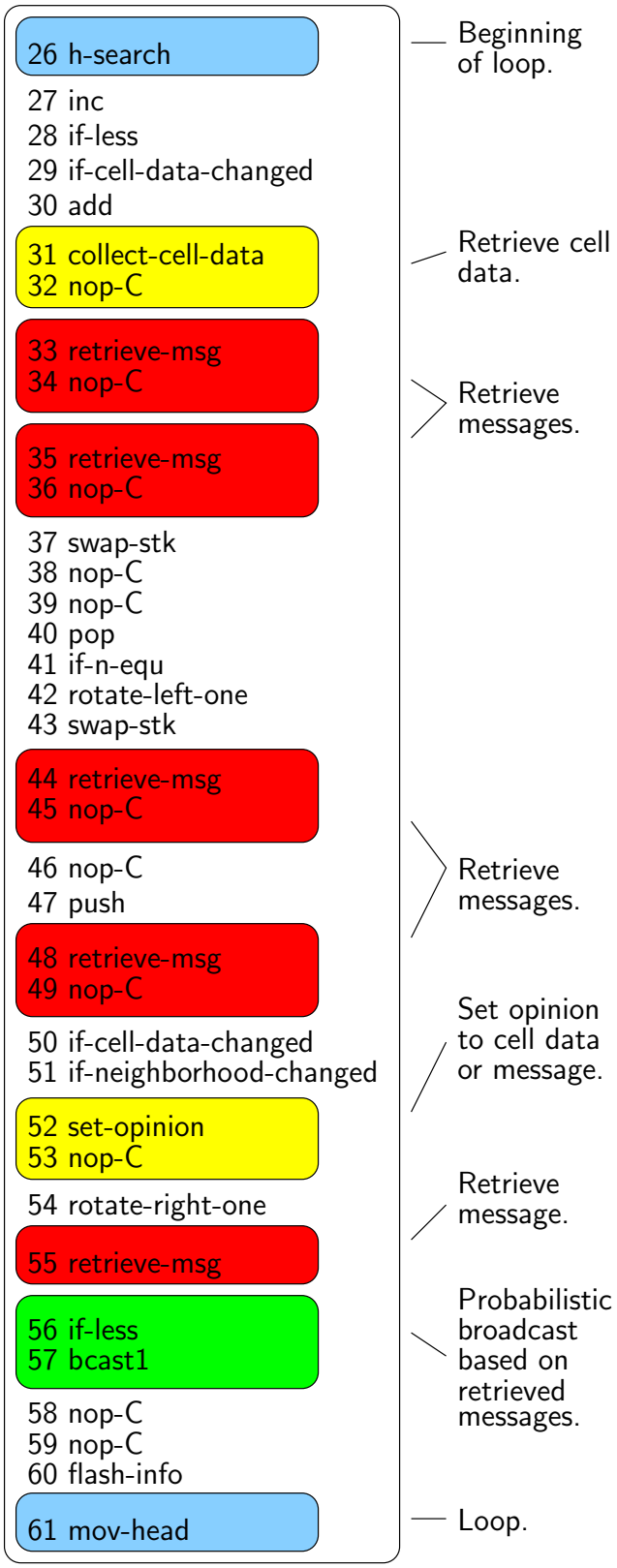


Figure 3.25: Annotated portion of the best-performing final dominant genome from the experiment described in Section 3.5.2.

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 synchroneity among organisms, a property that has been shown to increase system stability [119].

Algorithm 3.1 Evolved algorithm for solving the ICD.

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

```
    loop
 2:   CX ⇐ cellData
      (CX, AX) ⇐ retrieveMsg()
 4:   (CX, AX) ⇐ retrieveMsg()
      (CX, AX) ⇐ retrieveMsg()
 6:   (CX, AX) ⇐ retrieveMsg()
      setOpinion(CX)
 8:   (BX, CX) ⇐ retrieveMsg()
      if BX < CX then
10:     broadcast(CX, AX)
      end if
12: end loop
```

3.6 Effects of Population Structure

Having observed the evolution of both leader election and consensus under various population structures, including individual selection, group selection, and germlines, we now turn to a broader investigation of the effect of different biologically-inspired population structures on the evolution of consensus. The motivations for this study were two-fold: First, there is broad interest in using groups of evolved individuals to solve complex problems. These problems vary widely and include diagnosing malignancy in cancer [120], improving runtime

performance of multiobjective evolutionary algorithms [121], and controlling mobile sensor networks [54]. Second, we wished to discover the appropriate population structure for our remaining studies into the evolution of other distributed behaviors.

When discussing population structure, the two key differentiating aspects are the *level of selection* used (i.e., whether selection operates on individuals or groups) and the *genetic variation* present within the group (i.e., whether individuals within groups are genetically homogeneous or heterogeneous). Waibel, Keller, and Floreano provided an excellent overview of research in this area [103]. Their classification takes into account whether groups are homogeneous [122,66] or heterogeneous [123], and whether selection is performed at the level of the individual [66] or group [122, 123]. Correspondingly, they compare the performance of four types of groups: (1) homogeneous groups developed using individual selection; (2) homogeneous groups developed using group selection; (3) heterogeneous groups developed using individual selection, and (4) heterogeneous groups developed using group selection, on four cooperative tasks. Their results indicated that the genetic composition of groups and the level of selection used are critical factors for evolving groups that perform cooperative tasks, and that homogeneous groups tend to produce the most effective solutions. In this section, we focus on group-level selection and investigate the effect of intermediate levels of genetic variation on the evolution of solutions to the Iterated Consensus Dilemma.

3.6.1 Methods

Here we investigate six different treatments that modify deme- and population-level genetic variation. Our general approach is to adjust both the type of individual replication and also the presence of any gene flow among demes. All of the experiments were performed using the

same basic configuration as that described in Section 3.5, where we used the CompeteDemes framework to discover solutions to the Iterated Consensus Dilemma. The kinds of genetic variation explored here are summarized in Table 3.4, and described in more detail below.

Table 3.4: Types of genetic variation explored in our study of the effect of genetic variation on the evolution of consenus.

Treatment	Description
Homogeneous	Individuals within these demes reproduce using germlines. There is no gene flow among populations.
Asexual	Individuals within these demes reproduce using asexual replication. There is no gene flow among populations.
Sexual	Individuals within these demes reproduce using sexual replication. There is no gene flow among populations.
HGT	Individuals within these demes reproduce using asexual replication with horizontal gene transfer. There is no gene flow among populations. We experiment with seven different HGT treatments that vary the probability of an HGT event during self-replication from 0.1% to 80%.
Migration	Individuals within these demes reproduce using asexual replication. Gene flow occurs among demes as a result of migration. We experiment with ten different migration treatments that vary the probability that an offspring organism will be placed in a different deme from 0.6% to 100%.
Wilson	Individuals within these demes reproduce using asexual replication. Gene flow occurs immediately after deme competition as result of population shuffling.

Individual-level replication. At the individual level, organisms compete with each other for space in their environment and are responsible for their own replication (that is, the instructions for replication must be present in each organism's genome). The method by which an individual reproduces affects the amount of genetic variation present within the deme. Figure 3.26 depicts the four different individual replication methods used here: (1) germline, (2) asexual, (3) sexual, and (4) horizontal gene transfer.

For *germline replication*, we attach a germline, or heritable lineage, to each deme in the

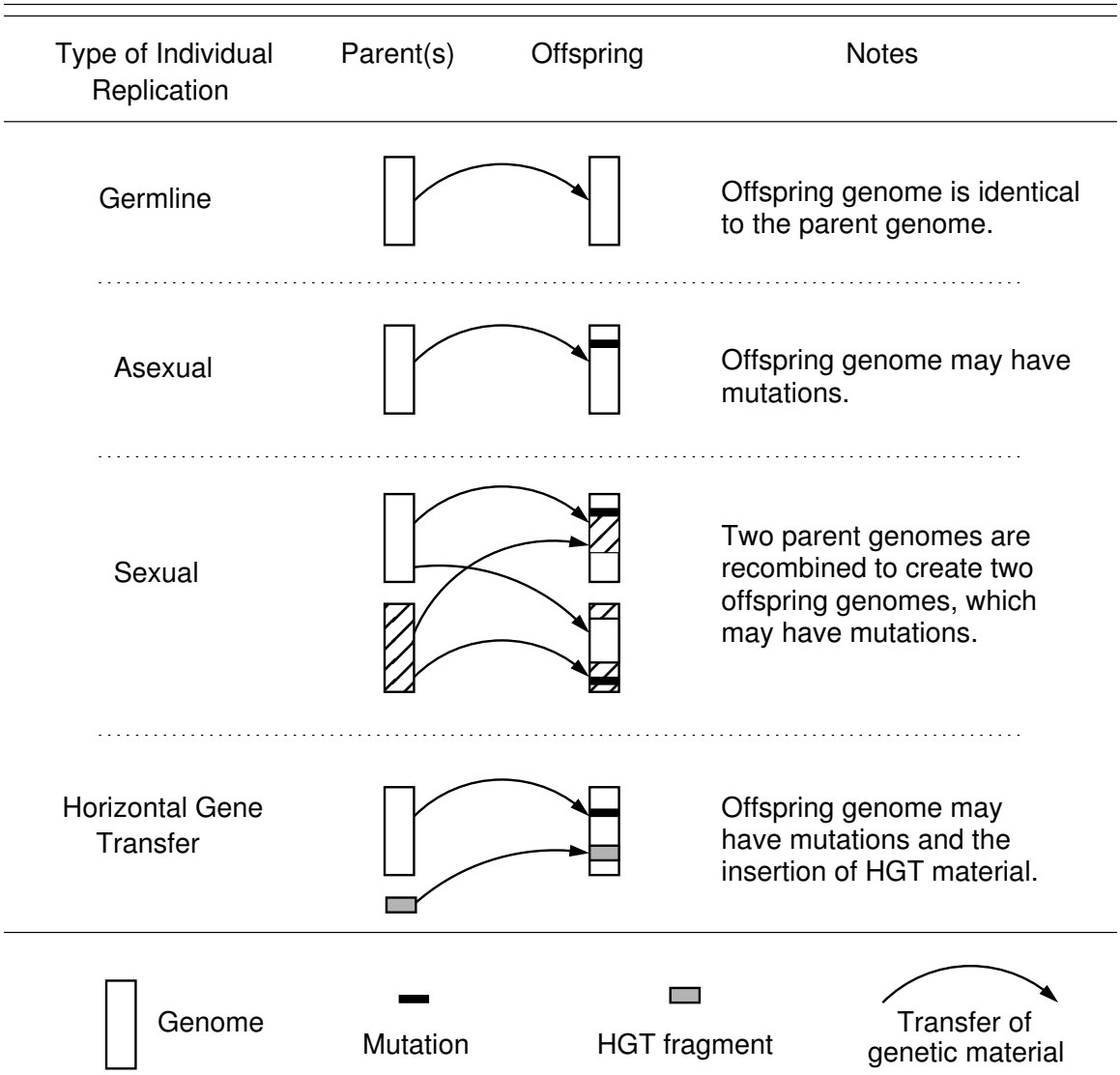


Figure 3.26: Depiction of the various types of individual replication used in this paper: germline reproduction, asexual reproduction, sexual reproduction, and HGT reproduction.

Avida population, as has previously been described. When an individual within such a deme replicates, it produces an identical offspring (Avida does provide options for these offspring to be mutated, but this feature is not used here). Mutations to the germline are introduced only when the deme replicates. This form of individual replication produces homogeneous (or clonal) demes.

For *asexual individual replication*, when an organism self-replicates, its genome is prob-

abilistically mutated to produce the offspring. This is the default individual replication method used within AVIDA [75]. In contrast, *sexual replication* mixes the genetic material of two parent organisms to produce two offspring. Specifically, when two organisms within the same deme replicate, their genomes are recombined via two-point crossover. The two resulting offspring genomes are them probabilistically mutated.

In biology, *horizontal gene transfer* (HGT) is a mechanism by which genetic material can spread among individual microorganisms that normally reproduce asexually [124, 125]. HGT is even known to permit the transfer of genetic material between different biological domains [126, 127]. Similar to the role that HGT plays in the rapid spread of antibiotic resistance in biology [125], it is possible that HGT in a digital form could contribute to the spread of beneficial innovations throughout a population of digital organisms. Within HGT treatments, organisms replicate asexually. During replication, a genome fragment selected from a random organism within the deme may be mutated into the offspring's genome, according to a predefined probability and segment matching criteria. This fragment then replaces the corresponding region in the offspring's genome that is nearest to the fragment, based on a substring match. This form of HGT approximates bacterial conjugation, where genetic material is transferred among normally asexual bacteria [128].

Algorithm 3.2 is pseudocode for the algorithm used in AVIDA to insert a horizontally-transferred genome fragment into a replicating organism's offspring. First, we test that an HGT event should occur based on a configurable probability (line 1). We then select a random genome fragment from a randomly selected organism in the same deme. This fragment is then compared to the offspring's genome using a substring match (lines 2-3), and the fragment replaces the most closely-matched region in the offspring genome (line 4).

Algorithm 3.2 HGT Insertion Algorithm

Require: *organism* is replicating.

```
    if P(HGT) then
2:      f ⇐ random(fragment)
        loc ⇐ substringMatch(f, offspring)
4:      offspring ⇐ hgtReplace(f, loc, offspring)
    end if
```

Gene flow among demes. One other aspect that affects genetic diversity within demes is *gene flow*, which is the movement of gametes, individuals, and populations throughout their environment [129]. In biological systems, while its impact may not be fully agreed upon [130], understanding gene flow has the potential to help us uncover disease-causing genes [131].

In general, offspring organisms are placed into the same deme as their parent and remain in these demes throughout their lifetime. Thus, for most of our treatments, there is no gene flow among demes. However, we also examined treatments that moved genetic material among groups, depicted in Figure 3.27. Specifically, we explored *migration*, where an organism's offspring is probabilistically placed into a different deme than its parent. Additionally, we examined Wilson's model for multilevel selection, which includes a specific dispersal stage where individuals migrate among demes [59]. In Wilson's model, groups evolve as under the asexual self-replication treatment. However, following deme competition (that is, after demes with high fitness have been replicated), all organisms throughout the population are randomly assigned to new demes ("shuffling" the population). While organisms that were members of successful demes are likely to be more prevalent, they may not be placed into demes with organisms that they have previously interacted with.

For all treatments, we configured Avida as in Section 3.5, with the exception that we require organisms to be self-replicating, even in the case of germlines, in order to normalize

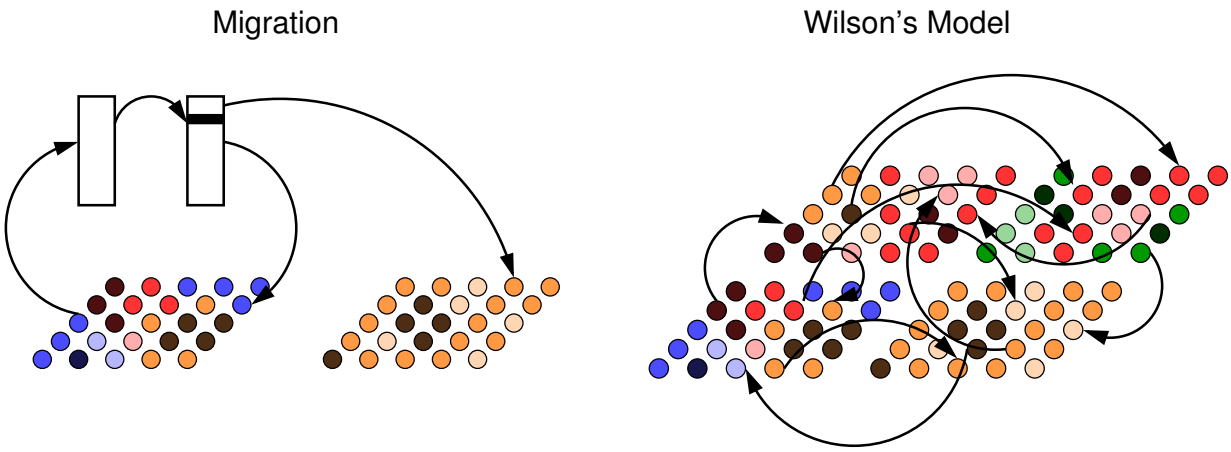


Figure 3.27: Depiction of migration (left) and Wilson's model (right), both of which serve to introduce gene flow among demes in AVIDA. In this figure, rectangles represent genomes, arrows represent gene flow, and colors within the demes represent different genotypes.

the different replication schemes across treatments. In non-germline treatments, we also scale mutation rates down by a factor of 25 to account for the larger effective population size. The reason for this is that, in the germline treatment, each deme has 25 copies of the same genome, and this single genome is probabilistically mutated when the deme is replicated. In contrast, each heterogeneous deme has 25 potentially different genomes that are mutated when organisms self-replicate. Thus, to enable a fair comparison among treatments, we decreased the mutation rate of organisms in heterogeneous demes. We also require that at least one individual within each deme self-replicate prior to the deme being replicated; if no individuals within a given deme self-replicate, then the deme is killed during the deme competition process.

3.6.2 Homogeneous Groups

While the study from Section 3.5 demonstrated the evolution of consensus in homogeneous groups, and in fact produced a novel algorithm for distributed consensus, it did so without

self-replicating organisms. As a baseline experiment, here we replicate the previous germline treatment with the additional requirement of self-replication.

Figure 3.28(a) plots grand mean deme fitness in units of number of consensus rounds achieved. As seen in this figure, homogeneous groups achieve 3 rounds of consensus on average, greater than the previous (non-self-replicating) treatment's average of 1.5. We suspect this difference is due to the presence of a *copy loop*, which is the series of instructions responsible for self-replication. The copy loop is easily adapted by evolution to perform other tasks. Figure 3.28(b) plots the average genetic variation within the entire population (error bars represent 95% confidence intervals constructed via 200-fold bootstrapping). Here, we define genetic variation as the *Levenshtein distance*, the minimum number of insertion, deletion, or substitution operations required to transform one genome into another. Specifically, population-level genetic variation is the mean Levenshtein distance of 5,000 pairs of organisms selected randomly without replacement, and each organism in the population is involved in one pair. This plot shows that after 200,000 updates, on average, 29 instructions must be changed to transform one genome into another. These results serve as a baseline for comparison for other treatments.

3.6.3 Within-group Variation

In this series of experiments we investigate various mechanisms for the mixing of genes within groups. Specifically, we consider three complementary treatments: *asexual*, where individuals within groups may undergo mutation during self-replication; *sexual*, where individuals are recombined during replication and then the resulting genomes may also undergo mutation; and *horizontal gene transfer* (HGT), where genome fragments from other organisms may

be integrated into offspring during (otherwise normal) asexual self-replication. In these and all non-homogeneous treatments, we have reduced copy, insertion, and deletion mutation rates by a factor of 25 relative to homogeneous groups to account for the change in effective population size. For all of these treatments, if a deme is replicated, all organisms living within the deme are copied to the offspring deme, and sterile demes (those in which no organism self-replicates) are not allowed to replicate.

Figure 3.29(a) plots grand mean deme fitness in terms of consensus rounds for the asexual and sexual treatments. Here we see that, on average, asexual groups achieve 1.2 consensus rounds, while sexual groups achieve 1.4 rounds. Interestingly, this result shows that heterogeneous groups have evolved to cooperate without enforced genetic homogeneity, although their results are not as strong as the homogeneous groups, which achieve 3 rounds of consensus on average. Figure 3.29(b) depicts the final number of consensus rounds achieved for rates of HGT varying from 0.001 to 0.8. As seen here, low rates (those less than 0.4) tend to result in more variation of performance, while high rates of HGT approach 1.4 consensus rounds, the same as was seen with sexual recombination. Intuitively, this result makes sense given high levels of HGT function similarly to sexual recombination. Overall, these results indicate that introducing genetic variation via sexual reproduction or HGT is somewhat disruptive to reaching consensus.

Figure 3.30 is a box plot depicting final population- and deme-level genetic variation for asexual, sexual, and HGT treatments. In general, we see population-level variation on the order of 10 instructions, while variation within demes is approximately 1 instruction. Deme-level genetic variation was calculated similarly to population-level variation, using 12 randomly selected pairs per deme. Compared to homogeneous groups, the sources of

within-group variation tested here exhibit less population-level diversity, while having only modestly increased variation within groups. Essentially, in order to cooperate, these demes have reduced their genetic variation to approach that of homogeneous groups.

3.6.4 Among-group variation

We now turn to treatments that address the mixing of individuals among groups. Specifically, here we investigate varying rates of migration as well as Wilson's model for group selection. In migration treatments, individuals are probabilistically migrated between demes upon replication, and we sample migration probabilities from 0.00625 to 1.0 (we note that at a migration rate of 1.0, all offspring are deposited in a different deme). Wilson's model is slightly different, in that individuals are not migrated between groups as a result of their replication, but the entire population is mixed at the start of each deme competition period.

Figure 3.31 depicts the final number of consensus rounds achieved for these different treatments. Migration treatments are shown as m.x, with the specific trailing number x indicating the migration rate, while Wilson's model is shown as wilson. Here we see significant variation in fitness among treatments with migration rates less than 0.2, and we see a decline in fitness as we increase migration rate from 0.4 to 1.0. In general, larger migration rates result in less cooperation, most likely due to increased competition within demes, and Wilson's model achieves approximately the same fitness as a migration rate of 0.4, which is unexpected given the extensive population mixing.

Figure 3.32 depicts final population- and deme-level genetic variation for different migration rates and Wilson's model. Here we see that, in general, as migration rates increase, both population- and deme-level variation increase. Moreover, compared to variation of

within-group treatments, migration treatments exhibit much greater genetic variation. This greater genetic variation appears to make it difficult for the demes to cooperate to perform consensus.

3.6.5 Discussion

We now turn to a discussion of the relationship of fitness to population- and deme-level variation, and a further analysis of these results.

Genetic variation and fitness. Figure 3.33 plots the relationship between mean deme- and population-level genetic variation for all treatments, where a larger point size corresponds to higher fitness. To reduce the impact of outliers on this figure, deme- and population-level edit distances were log-transformed prior to the calculation of the mean; this figure depicts the untransformed mean. Based on this figure, we can make a number of observations. First, not only is it clear that homogeneous groups excel at solving the consensus problem, they also have a unique combination of population and deme variation. Specifically, homogeneous groups (germ) have low (zero) deme-level variation, while experiencing moderately high population-level variation. Second, we see that within-group mixing (sexual replication [sex], HGT [h.001-h8]) are generally homogenizing forces at both the group and population levels. Intuitively, this make sense: beneficial mutations that fixate in a deme will quickly spread throughout the population via deme competition. We conclude that the reduction in fitness compared to homogeneous groups is then a result of competition among members within a group. Third, we see that, except for very low rates, among group variation (migration [m.006-m1.0] and wilson [wilson]) not only introduces a fitness penalty, but also greatly increases both population and deme variation. This trend indicates that it is not only better

to err on the side of too little among-group mixing, but that it can be eliminated entirely with little penalty. Among the biologically-inspired approaches to genetic variation that we investigated, homogeneous groups occupy a unique space: low within-group variation, but large among-group variation.

Group cohesion. One hypothesis for the relationship among the different treatments tested here is that *group cohesion* contributes to the evolution of cooperative behavior. In this case, we define group cohesion as whether individuals within a group remain with each other during evolution, or if they are likely to change groups. For example, asexual and sexual treatments both have high group cohesion, while migration treatments have group cohesion that depends on migration rate. Intuitively, groups with high cohesion would seem more likely to evolve cooperative strategies, as individuals within the groups are exposed to each other over evolutionary time. To test this hypothesis, we grouped treatments by whether or not cohesion was maintained. Cohesive groups corresponded to asexual, sexual, and HGT treatments, while non-cohesive groups corresponded to migration and Wilson's model treatments. Homogeneous groups are clearly cohesive, and thus were left out of this analysis. We found that the fitness values of cohesive groups were statistically significantly higher than the fitness of non-cohesive groups (Wilcoxon rank sum test, $p = 7.5 \times 10^{-26}$). This result suggests that group cohesion may be a significant factor for the evolution of cooperation.

Models of diversity. While the graphical representations of our results indicate that diversity may be a good predictor of fitness, we also created several linear models of the relationship between genetic variation, fitness, and treatment type. These models provide a

way to quantify the relationship among various aspects of diversity and fitness. Specifically, while Figure 3.33 appears to indicate that treatment type (e.g., germline) is the primary indicator of success, models of the relationship among genetic variation, treatment, and fitness enable us to determine the key indicators of success qualitatively. All analyses were performed using R version 2.11.1 (R Development Core Team 2007).

Table 3.5 summarizes these models. For these models, we consider either average fitness or maximum fitness to be the dependent variable we are trying to predict. Specifically, models A and AT predict average fitness, while models M and MT predict maximum fitness. We then investigate whether population-level genetic variation (*pop*) specified in terms of mean Levenshtein distance, deme-level genetic variation (*deme*) specified in terms of mean Levenshtein distance, and treatment type, e.g., homogeneous, asexual, sexual, expressed as a categorial factor (*treat*) can be used to explain the observed fitness values. We include treatment type as a potential factor for models AT and MT because it is possible that the method by which the genetic diversity is produced also affects fitness. All models assume a normal distribution of fitness values. For these models, we used R to estimate the β parameters and σ^2, which represents the error. Table 3.6 summarizes the beta values for these models, as well as an indication of significance of each factor found using Anova. The values for $beta_3$ are not shown, as they reference treatment category. For these models, $beta_1$ (deme-level variation) and $beta_3$ (treatment category) are always statistically significant ($p < 0.001$). $Beta_2$ (population-level variation) is only statistically significant when predicting maximum fitness.

Next, we ranked the models based on their respective Akaike Information Criterion (AIC) [132]. AIC measures how well the model fits the data given the number of pre-

Table 3.5: Candidate models and Akaike Information Criteria (AIC) results for average and maximum fitness. Predictors are the final population (*pop*) and deme (*deme*) variation averages, as well as treatment category (*treat*).

Model	K_i	AIC	Δ_i	DoF	Rank
MT (maximum fitness, with treatment category): $maxFit \sim N(\beta_0 + \beta_1 deme + \beta_2 pop + \beta_3 treat,\ \sigma^2)$	3	10620.25		10	1
AT (average fitness, with treatment category): $avgFit \sim N(\beta_0 + \beta_1 deme + \beta_2 pop + \beta_3 treat,\ \sigma^2)$	3	11098.57	478.32	4	2
M (maximum fitness): $maxFit \sim N(\beta_0 + \beta_1 deme + \beta_2 pop,\ \sigma^2)$	2	12090.60	1470.35	10	3
A (average fitness): $avgFit \sim N(\beta_0 + \beta_1 deme + \beta_2 pop,\ \sigma^2)$	2	12321.16	1700.91	4	4

Table 3.6: Beta values and significance for the fitted versions of the four linear regression models. Within the table, *** p $<$ 0.001, ** p $<$ 0.01, * p $<$ 0.05.

Model	β_1	β_2	β_3
A	-43.714***	5.218	
M	-20.273***	5.057*	
AT	-4.475***	-6.652	***
MT	-1.7339***	-0.8996***	***

dictive factors it uses. A lower AIC value indicates a better fit. In this case, the AIC values show that including treatment category improved explanatory power. This indicates that the type of variation provided by the treatment has a significant effect on fitness. In general, the *beta* values for these models indicate that deme diversity is negatively correlated with both average and maximum fitness (negative $beta_1$), providing further evidence that homogeneous groups are more effective at cooperative behavior. One explanation for this result is that greater deme diversity increases competition within demes and thus inhibits the evolution of cooperation. Interestingly, population diversity ($beta_2$) is only correlated with maximum fitness (models M and MT). The direction of the effect of population diversity is negative when treatment category is included (model MT), but reversed when treatment category is removed as a factor (model M). Thus, the population diversity may enable the exploration of additional areas of the search space when group membership is fixed, i.e., for treatments in which demes remain cohesive. However, population diversity can be a divisive force when group cohesiveness is not maintained.

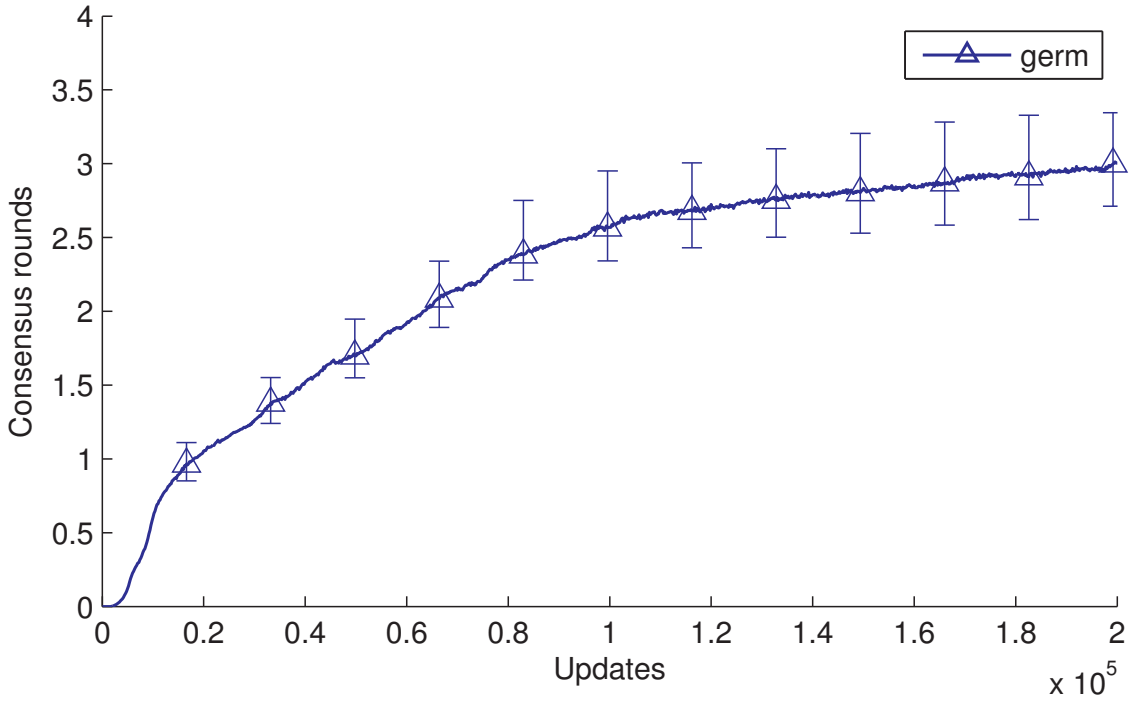
3.7 Conclusion

In this chapter, we have presented the results of a series of studies of data-driven behavior. In our earlier studies of leader election, we first demonstrated the evolution of collective behavior in single populations. To do so, we applied selective pressures by way of Avida tasks on individual organisms. Notably, these tasks required knowledge of the global state to function. We next explored the use of multilevel selection to evolve leader election in heterogeneous groups of self-replicating organisms. In this case, the evolved behaviors depended on an ongoing process of mutation within demes. Thus, while we were able to evolve leader election

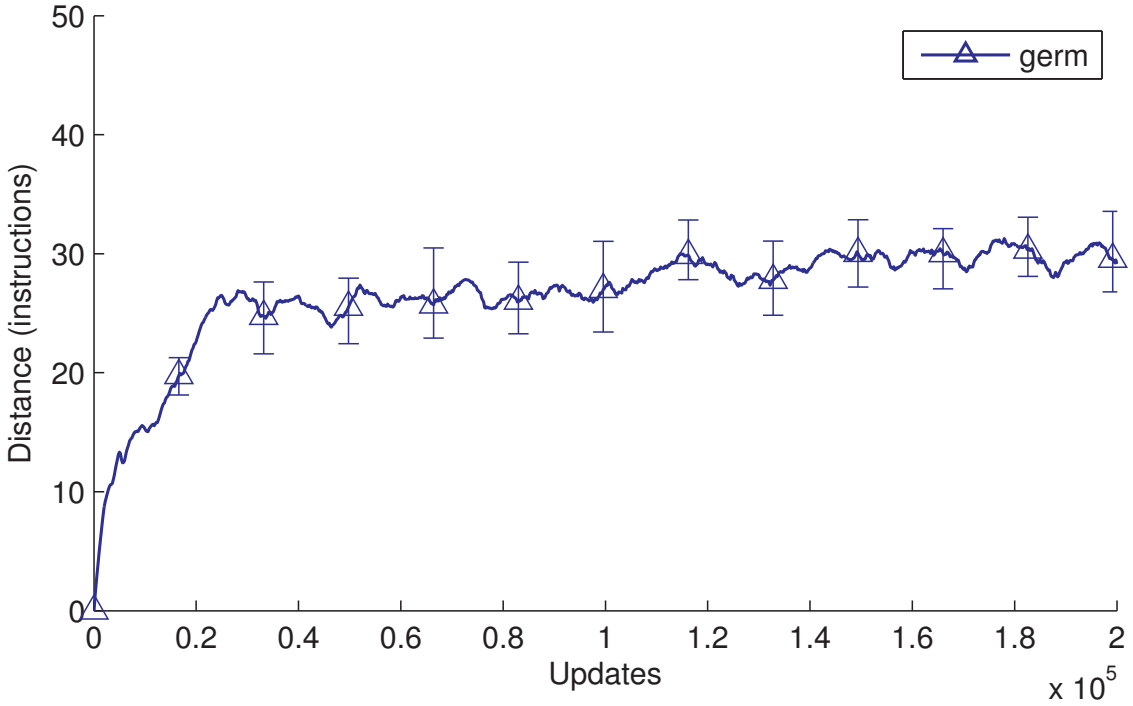
without specifying selective pressures that acted on individuals, the behaviors were not suitable for real systems.

We next turned to a study of a general form of consensus, and showed how homogeneous groups of digital organisms could evolve to perform consensus. This approach, in fact, produced a novel algorithm for probabilistic consensus – The first known instance of a distributed algorithm discovered by digital evolution.

Finally, we also presented results showing that homogeneous groups, out of a wide variety of biologically-inspired population structures, were most effective at evolving solutions to consensus. Due to this result, which is consistent with other research that addresses population structure [103], the remaining studies in this dissertation focus on homogeneous groups. Data-driven behaviors are but one component of modern distributed systems, and we now turn to a discussion of temporal behavior, specifically synchronization and desynchronization, which are both important parts of many distributed algorithms.

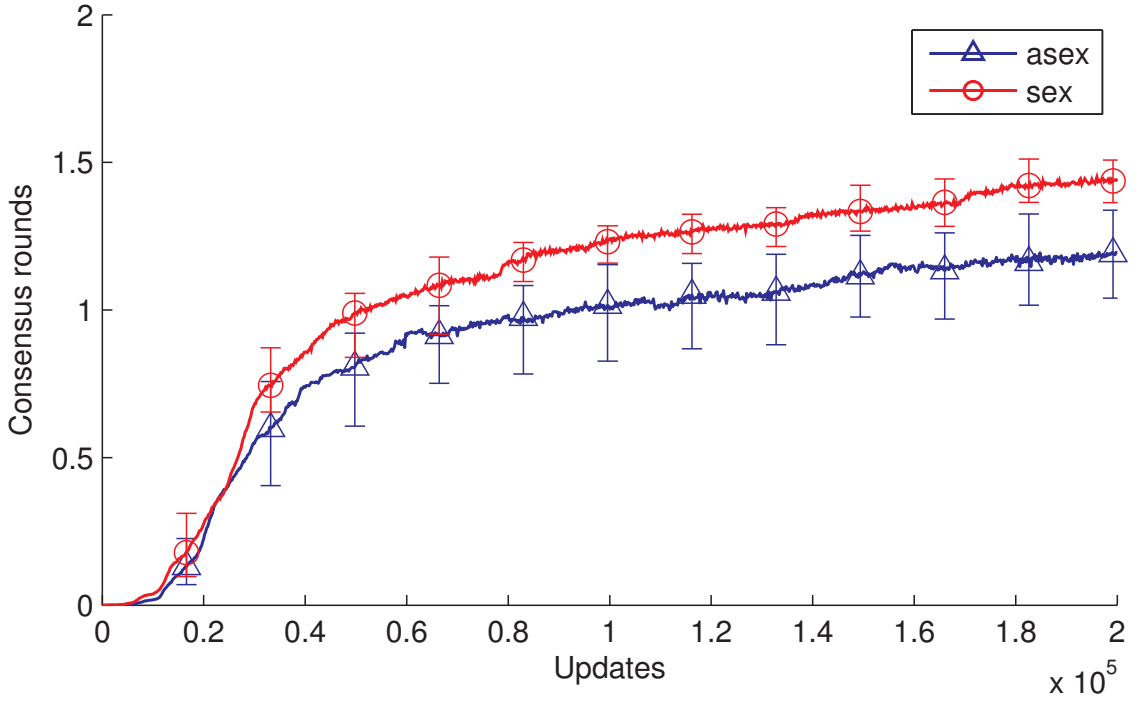


(a) Rounds of consensus achieved via homogeneous groups.

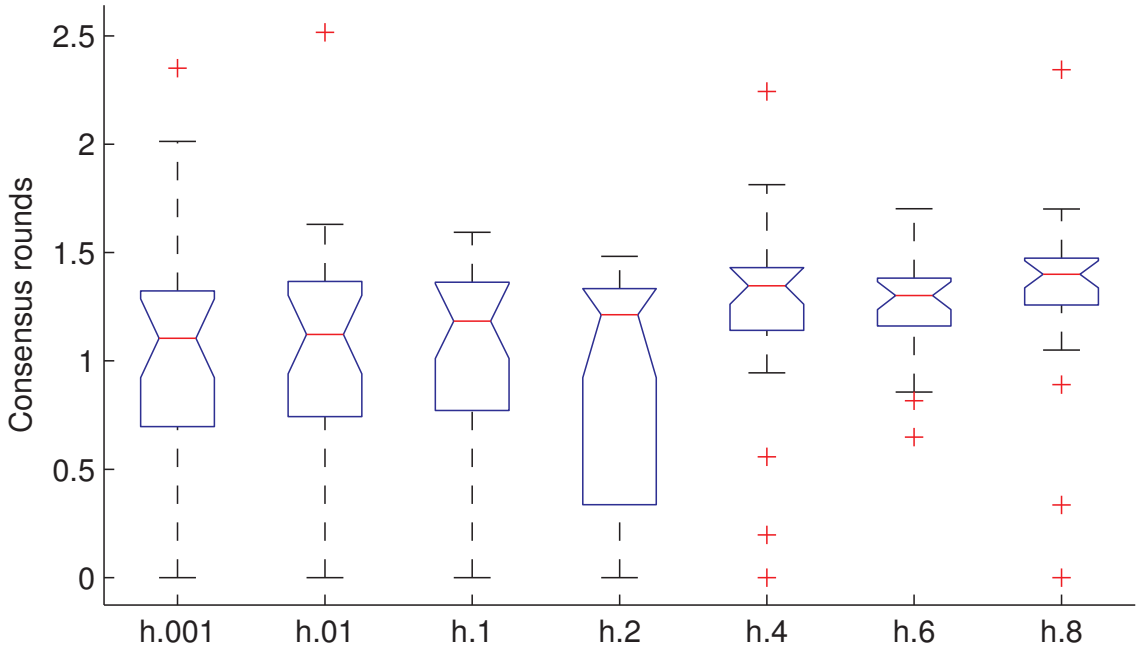


(b) Average population-level genetic variation.

Figure 3.28: Grand mean deme fitness in units of consensus rounds (left), and average genetic variation within the population (right). Demes achieve three rounds of consensus on average, with an average Levenshtein distance of 29 instructions between genomes.

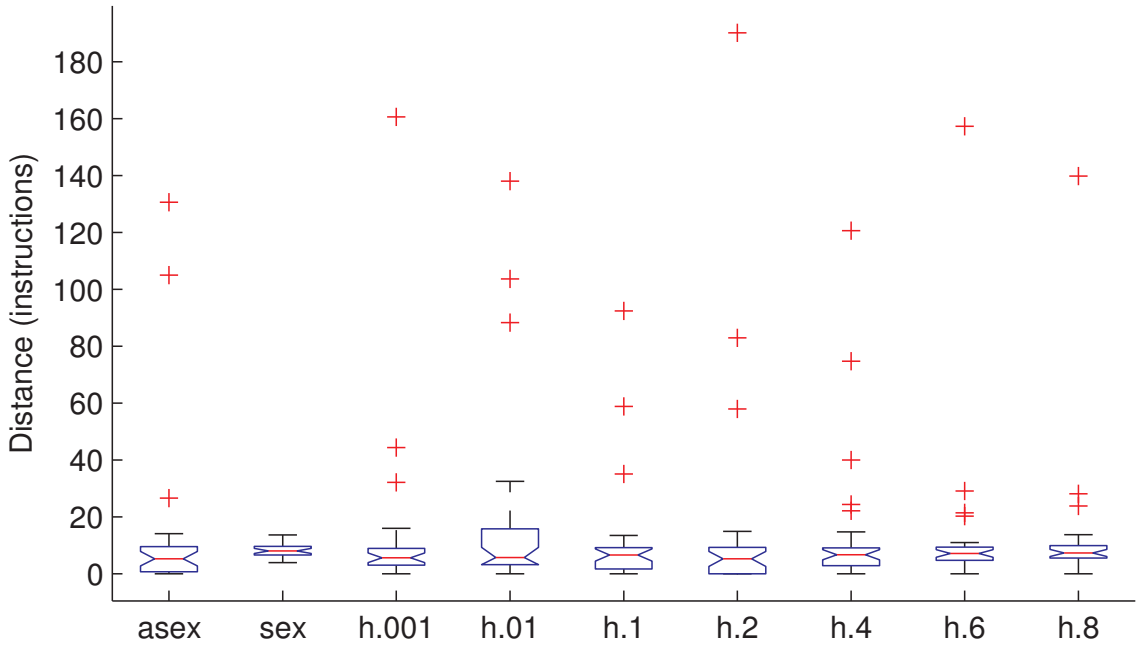


(a) Rounds of consensus achieved via asexual and sexual replication.

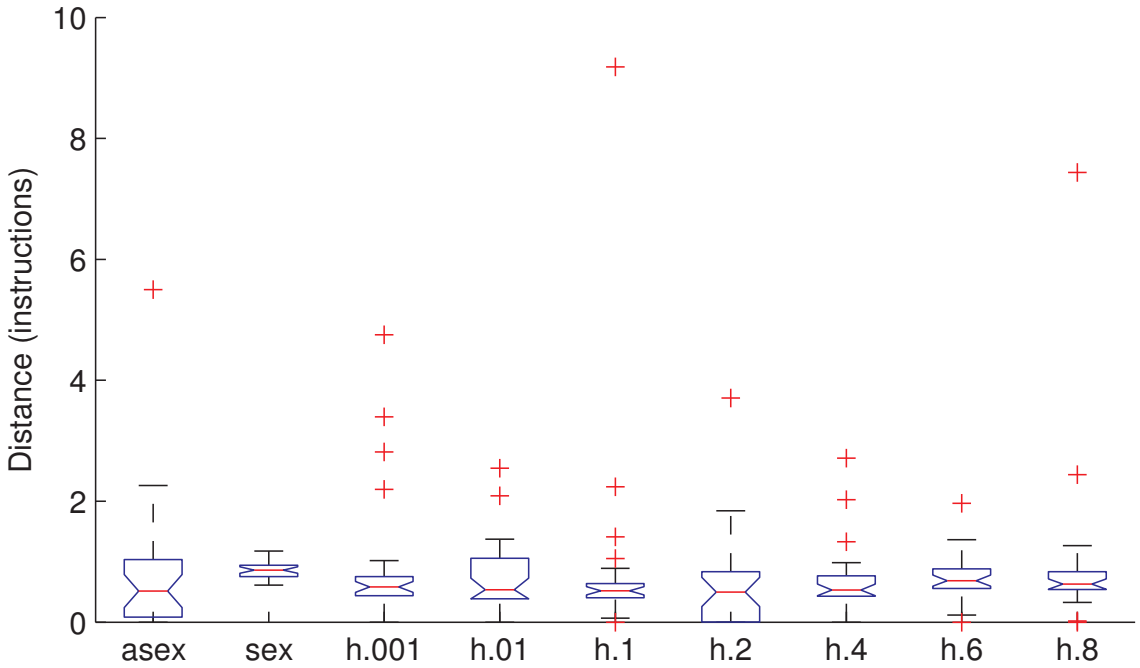


(b) Rounds of consensus achieved via different rates of HGT.

Figure 3.29: Grand mean deme fitness in units of consensus rounds for asexual and sexual replication (left), and varying rates of horizontal gene transfer (right).



(a) Average population-level genetic variation.



(b) Average deme-level genetic variation. To better show the relationship among treatments, the following outliers are not shown: Treatment asex at 117.37; h.01 at 68.18 and 125.66; and h.4 at 79.27.

Figure 3.30: Genetic variation at the population- and deme-level across asexual, sexual, and HGT treatments.

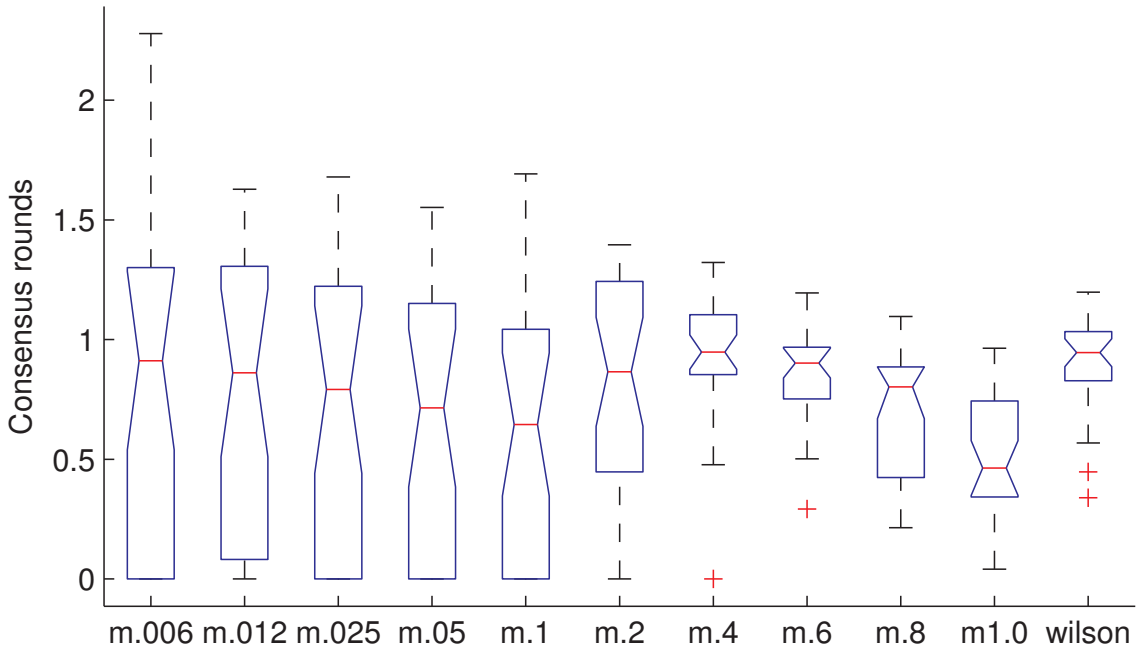
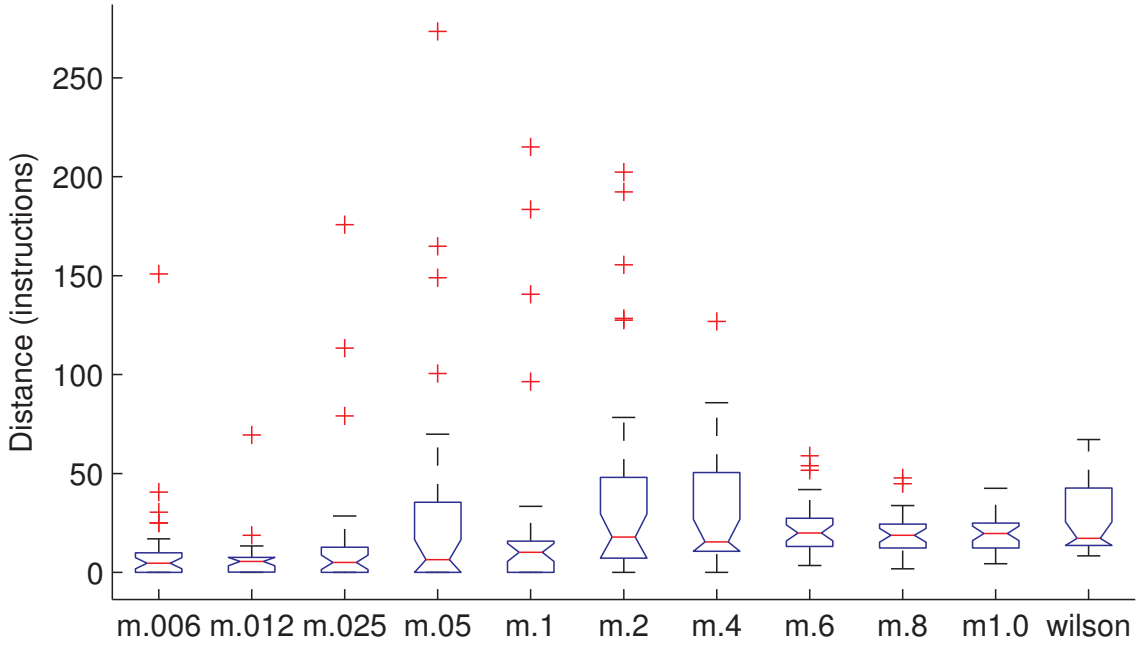
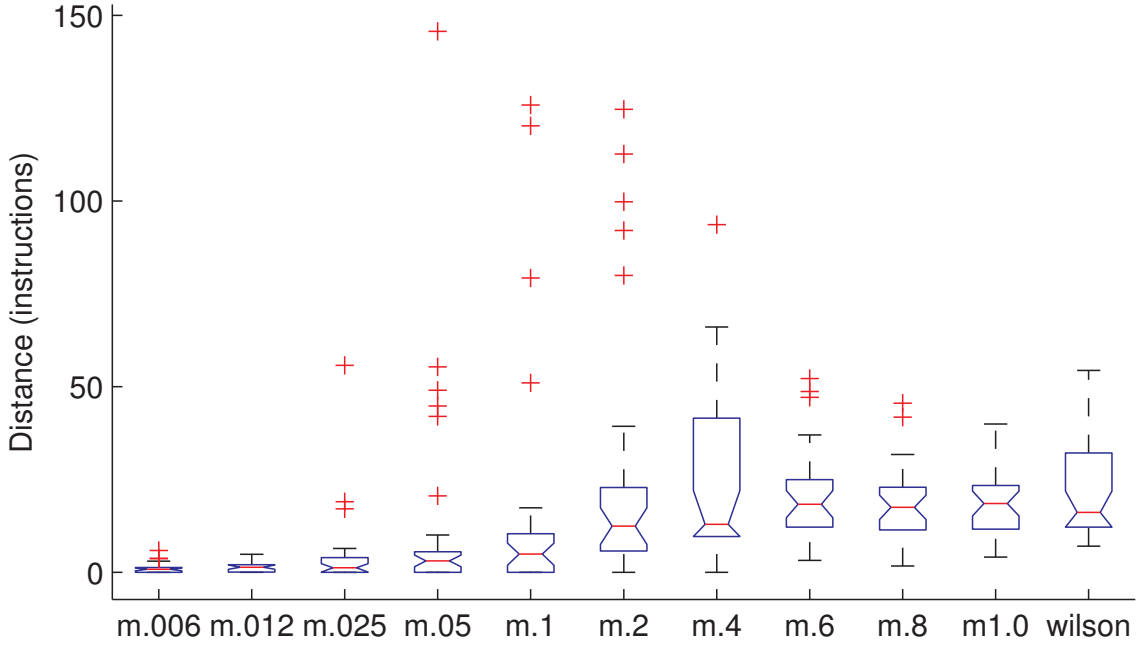


Figure 3.31: Grand mean deme fitness in units of consensus rounds for varying rates of migration and Wilson's model for group selection.



(a) Average population-level genetic variation.



(b) Average deme-level genetic variation.

Figure 3.32: Genetic variation at the population- and deme-level across varied migration rates and Wilson's model for group selection.

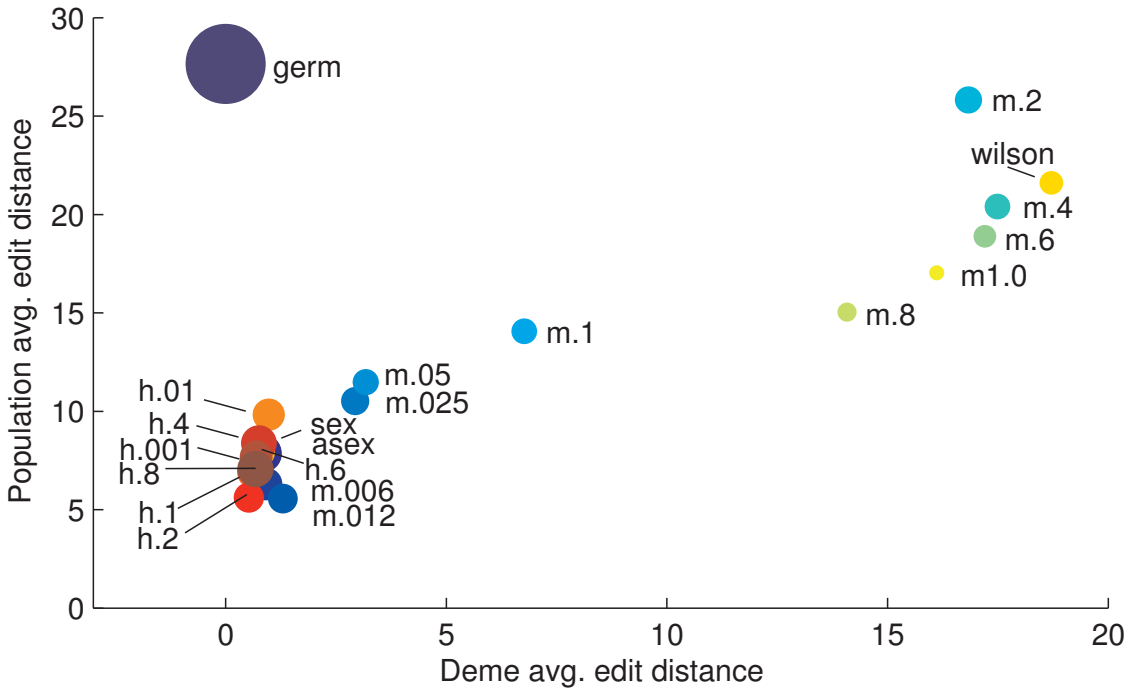


Figure 3.33: Average population- vs. deme-level genetic variation across all experimental treatments from this study. Homogeneous groups occupy a unique position in this space: high population-level variation, with low deme-level variation.

Chapter 4

Evolving Temporal Behavior: (De-)Synchronization

Temporal behaviors are based in some way on the passage of time and are found in many different species. Circadian rhythms, for example, are exhibited not only by mammals, but also plants and bacteria [133]. In addition, many species exhibit synchronous behaviors. For example, honey bees synchronize their activity cycles [134], fiddler crabs synchronously wave their oversized claw [135], and ants synchronize their alarm drumming [7]. Humans are also known to exhibit synchrony of neuronal activity within the cerebral cortex [136], as well as during social coordination [137]. One of the more striking examples of synchrony in the natural world is the coordinated flashing of male fireflies. In some parts of Southeast Asia, these fireflies synchronize their flashes to a common period and phase over a distance of many miles [36].

Temporal behaviors are also important in computational settings. For example, two topics of recent interest in distributed systems are heartbeat synchronization and desyn-

chronization. In heartbeat synchronization, nodes in a distributed system are required to generate periodic heartbeat signals at approximately the same time [33]. In contrast, *desynchronization* is where nodes with the same period are to distribute (stagger) their signals in phase-space, similar to round-robin scheduling [49]. Heartbeat synchronization is a common part of communication protocols that require restarts [45], and desynchronization can play an important role in medium access control, fair scheduling, and energy efficiency [49].

Recent bio-inspired studies of temporal behaviors include the work of Babaolgu [33], Patel [49], and Christensen [138], whose research into synchronization and desynchronization were in part inspired by the synchronization of firefly flashes. Their proposed algorithms were based on the mathematical modeling of pulse-coupled oscillators by Mirollo and Strogatz [48], as well as specific models of firefly synchronization by Ermentrout [47]. Babaoglu and colleagues designed a protocol for heartbeat synchronization that was based on the Ermentrout model, and showed that this mechanism was remarkably effective at synchronization, even in the presence of message loss. Patel and colleagues described two different protocols for desynchronization, one based on particle diffusion and another based on firefly synchronization, while Christensen and colleagues designed a series of synchronization algorithms for robots, and used the lack of synchronization to detect faults.

A recent study of the evolution of synchronization for robotic swarms was conducted by Trianni and Nolfi [139]. In that study, a neural network controller was evolved to synchronize the movements of robots within an enclosed arena. The robots were able to sense a symmetric gradient painted on the floor of the arena, and could communicate globally via a single audible tone. Our objective in the study described here was different, in that we sought to determine whether (and how) populations of digital organisms can evolve both syn-

chronization and desynchronization behaviors when provided with primitives akin to those of fireflies. Specifically, the organisms had no environmental cues and were only capable of local communication.

Our studies: Similar to how fireflies have evolved to synchronize their flashes, in this chapter we use digital evolution to evolve digital organisms that both synchronize and desynchronize their virtual "flashes" in response to different selection pressures. First, we demonstrate the evolution of synchronization and desynchronization behavior. Second, we investigate whether these behaviors are resilient to failures in the transmission of virtual flashes. Third, we perform an in-depth analysis of the genomes responsible for these behaviors, and in the case of synchronization, find that the evolved behavior uses a strategy remarkably similar to models of firefly synchronization in biology. Finally, these results demonstrate that digital evolution offers a promising approach to the design of temporal behaviors for distributed computing systems.

4.1 Methods

To evolve synchronization and desynchronization, we extended Avida with a small set of instructions that enable the transmission and reception of virtual flashes. For evolving cooperative behavior, we made use of digital germlines and the CompeteDemes framework, and defined fitness functions that rewarded groups of digital organisms for exhibiting synchronization and desynchronization behaviors.

Instructions. We implemented six new virtual CPU instructions for this study, which are summarized in Table 4.1. Each of these instructions was either inspired by observations of

fireflies (e.g., flash) or was implemented to provide digital organisms with information not previously available (e.g., flash-info). When an organism executes the flash instruction, a flash message is broadcast to each of its neighbors, and information about this message is automatically stored in a special register that can be accessed only via the if-recvd-flash, flash-info, and flash-info-b instructions. The if-recvd-flash instruction conditionally executes the next instruction in the organism's genome if the caller had received a flash message since its birth or the last execution of the hard-reset instruction (described below). The flash-info instruction is used to determine if the organism has ever received a flash, and if so, sets a register to the elapsed time (in number of virtual CPU cycles) since the last flash message had been received. Similarly, the flash-info-b instruction is used to determine if the organism has ever received a flash, but does not store the time at which that flash was received. The hard-reset instruction resets the state of the organism to its initial state (as if the organism were just born). Finally, the get-cycles instruction retrieves the number of virtual CPU cycles since the organism was born or last issued a hard-reset, whichever is most recent. We emphasize that these instructions are simply *available* to the evolutionary process. Whether they are incorporated into a digital organism's genome is a function of mutation and selection.

Configuration. We configured AVIDA to use the CompeteDemes process outlined in Section 2.3.3, with specific configuration values summarized in Table 4.2. We used 400 demes, each with a 5×5 torus topology, and each deme was configured to use a germline. The relatively small size of each deme was selected primarily for practical (i.e., computation time) purposes. The different mutation rates used correspond to the default values for AVIDA. To improve data granularity, we also adjusted the *time slice* down from a default of 30 virtual

Table 4.1: New instructions implemented for this study. All instructions are equally likely to be selected as targets for mutation.

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

CPU cycles per update to 5 cycles per update. We performed 30 trials of each experiment, except where noted.

In order to control the initial conditions for each deme, we configured Avida to disallow self-replication. For the synchronization experiments we then configured an event, Random-InjectGerm, to probabilistically (50%) insert an organism with the germline's genome into an empty cell in each deme every update, thus ensuring that the organisms in a deme were not initialized to a synchronized state. Similarly, in the desynchronization experiments we initialized each deme to be completely filled with synchronized organisms built from the germline's genome. These different forms of initialization ensured that demes were not synchronized (or desynchronized) due to artifacts of Avida. We used a genome that contained 99 nop-C instructions followed by single flash as the default ancestor for both studies.

Table 4.2: AVIDA configurations used for the synchronization and desynchronization experiments described in this study.

Configuration	Value
Trials per experiment	30, except where noted
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, constant-time scheduler
CompeteDemes	Compete all demes using fitness-proportional selection (periodic, every 200 updates).
RandomInjectGerm	50% probability to inject a copy of the latest germ into an empty cell in each deme (periodic, every update).

4.2 Evolution of Synchronization

For this study, we examined heartbeat synchronization [33], where we evolve digital organisms that are synchronized to within a single update, or 5 virtual CPU cycles, of each other. We initialized each deme to be in a state of desynchronization by starting from a single organism, and inserted exactly one new organism with a 50% probability each update. We configured the CompeteDemes framework to compete all 400 demes with each other every 200 updates, and the fitness function used was based on the execution timing of the flash instruction within each deme. Within each deme we monitored both the number of unique organisms that executed the flash instruction, as well as the absolute number of flash instructions that were issued.

Fitness function. We defined a fitness function that rewarded demes whose constituent organisms each executed the flash instruction, and then further rewarded demes for increasing the difference between the maximum number of flashes and the mean number of flashes.

Specifically, the fitness function F used for synchronization was defined as:

$$F = \begin{cases} 1 + U & \text{if } U < S \\ 1 + U + (max(C) - mean(C))^2 & \text{if } U = S \end{cases} \tag{4.1}$$

where U is the number of unique organisms that issued a flash, S is the number of organisms in the deme (or, 25 for these experiments), $max(C)$ is the maximum number of flashes during any single update, and $mean(C)$ is the mean number of flashes per update. This fitness function thus rewards demes whose constituent organisms all issue the flash instruction, and then further rewards for increasing flash synchronization. Each of $max(C)$, $mean(C)$, and U were calculated from the final 50 updates of each CompeteDemes period to allow organisms a time (the first 150 updates) during which they may issue flash instructions without adversely affecting the deme's fitness. This approach grants evolution a great degree of flexibility in the kinds of strategies that may evolve. Maximum fitness is achieved when all organisms flash exactly once, at approximately the same time, during the final 50 updates of a competition period. We note that synchronization did not evolve when fitness was calculated over the entire 200 update period.

Treatments. To better understand some of the factors that influenced the evolution of synchronization, we conducted three different treatments of the synchronization experiment. The *flash-info* treatment used a 20-cycle cost for execution of the flash instruction (all others required only a single cycle). The *zero-cost* treatment modified the flash-info treatment by reducing the cost of executing the flash instruction to one cycle, making the flash instruction no more expensive to execute than any other instruction in the genome. Both the flash and zero-cost treatments were allowed to use the flash-info instruction. Finally, the *no-timer*

treatment modified the flash-info treatment by replacing the flash-info instruction with flash-info-b, which prevented organisms from accessing timing information related to the reception of flash messages.

4.2.1 Experimental Results

Figure 4.1 plots the grand mean number of coincident flashes over all 30 Avida trials, for each of these three different treatments. Here we see that after 100,000 updates, the mean difference between maximum and mean flash counts approaches 20 in all three treatments. In other words, after 500 generations (100,000 updates / 200 updates per CompeteDemes period) organisms within the average deme in each treatment have synchronized with each other, and we see that this behavior does not depend on instruction cost or knowledge of the exact timing of message reception. We emphasize that this figure shows the *average* results; a number of demes achieved perfect synchronization, as shown in Figure 4.2. We note that the final results of these treatments are not statistically significantly different from each other (Kruskal-Wallis multiple comparison, test for significance performed at the $p < 0.05$ level; standard deviation at final update: $2.09, 2.33, 2.27$).

Figure 4.3 plots the grand minimum number of coincident flashes over all trials (that is, the *least* fit of all demes). Although this figure does little for evaluating the performance of the evolved solutions, it does grant insight into the evolutionary process that gave rise to synchronization. Specifically, here we see that the zero-cost treatment has the highest fitness until approximately update 95,000, when the performance of the flash-info treatment improves. This result indicates that having an instruction cost for issuing a flash hinders the evolvability of synchronization. We also see that the no-timer treatment performed worst for

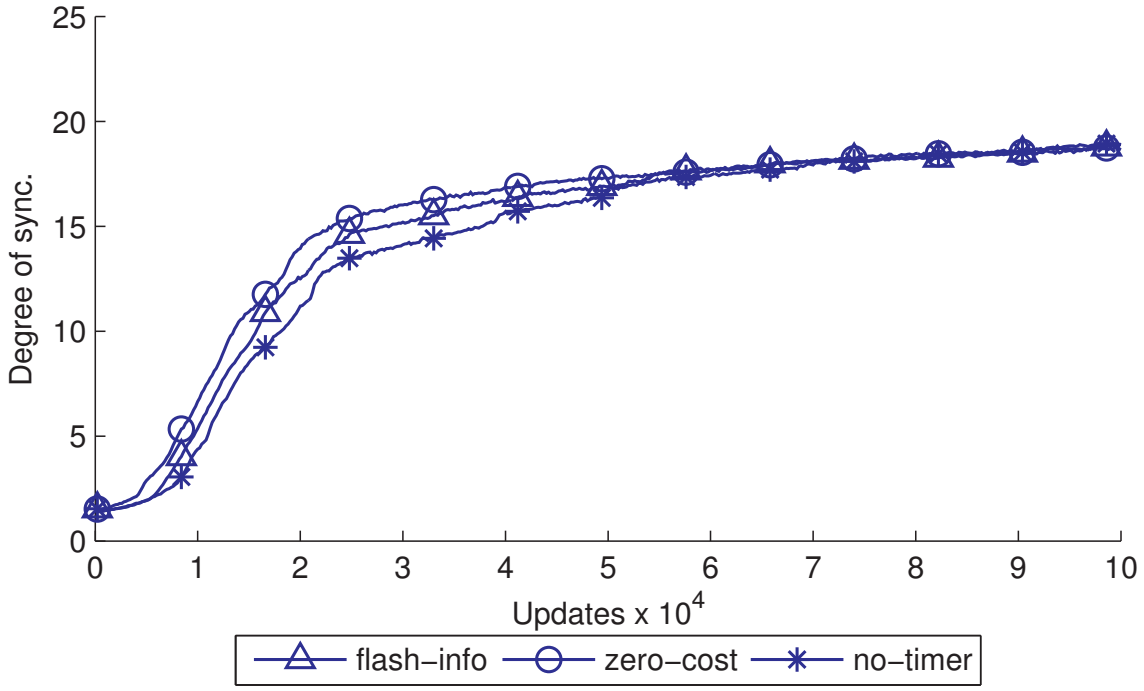


Figure 4.1: Evolution of synchronization using a fitness function that rewards for synchronization in execution of the flash instruction, under three different treatments.

approximately 70,000 updates, indicating that the availability of message timing information improved the evolvability of synchronization.

To determine if the evolution of synchronization is resilient to lost flash messages, we next examined the effect of flash loss rate on the evolution of synchronization. Specifically, we varied the flash loss rate from 0% to 80%, where we defined a flash "loss" as a flash message that is lost in sending, that is, it is not received by any of its neighbors. Each of these treatments used a unit-cost flash instruction. Figure 4.4 plots the number of coincident flashes versus time averaged over each deme in 10 trials, iterated over different flash loss rates from 0%-80% under the synchronization treatment. Here we see that higher loss rates inhibit the evolution of synchronization behavior, although we note that evolution was still able to discover solutions comparable to the 0% loss rate case at loss rates up to 20%. Statistically significant differences of the final results were found between the $\{0\%, 10\%, 20\%\}$

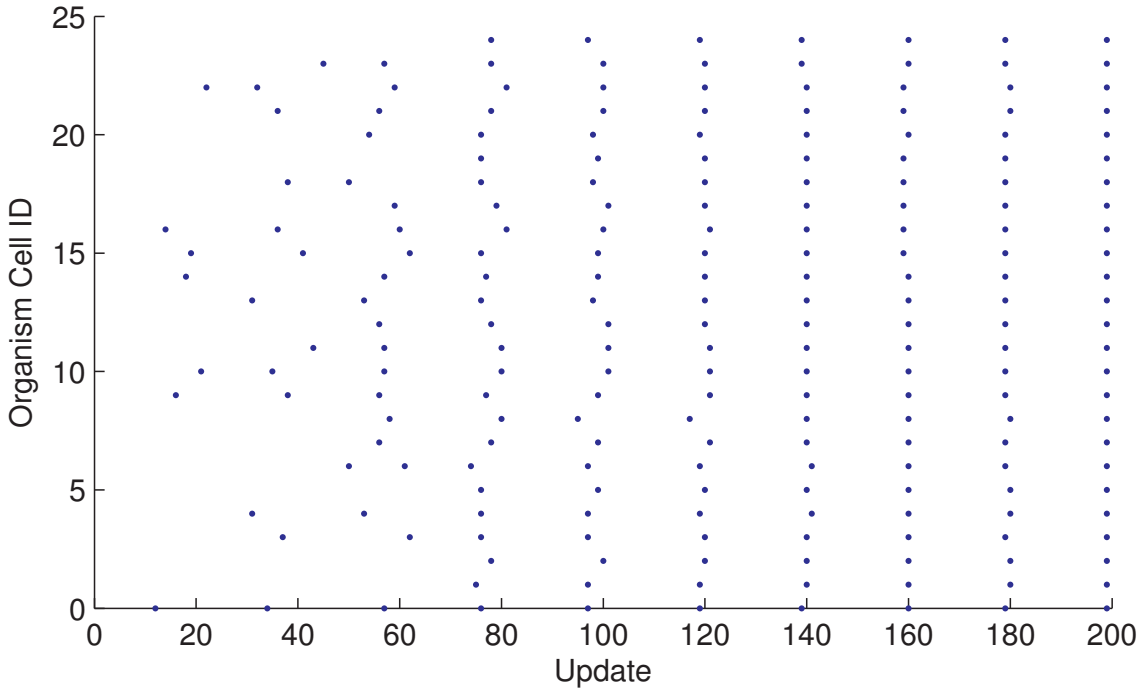


Figure 4.2: Detail of evolved synchronization behavior. Organisms initially flash asynchronously, but gradually synchronize with each other.

and $\{40\%, 80\%\}$ treatment groups, but no statistically significant differences were found within these groups (Kruskal-Wallis multiple comparison, test for significance performed at the $p < 0.01$ level).

Figure 4.2 is a detail of the behavior of a single deme containing 25 copies of the *dominant* (that is, most common) evolved genome from the end of the *flash-info* trials. Each point in this plot represents the execution of the flash instruction by a particular organism. Here we see runtime synchronization, where the 25 individuals within the deme have evolved to synchronize their flashes at an identical phase and period within 200 updates. Of note is that evolution has solved *two* different problems: the period that organisms will synchronize to, and the behavior that brings them into synchronization. Here, the organisms within this deme have synchronized to a period of 19 updates. Different demes within this same trial, as well as those in different trials, may well synchronize to a different period. A video showing

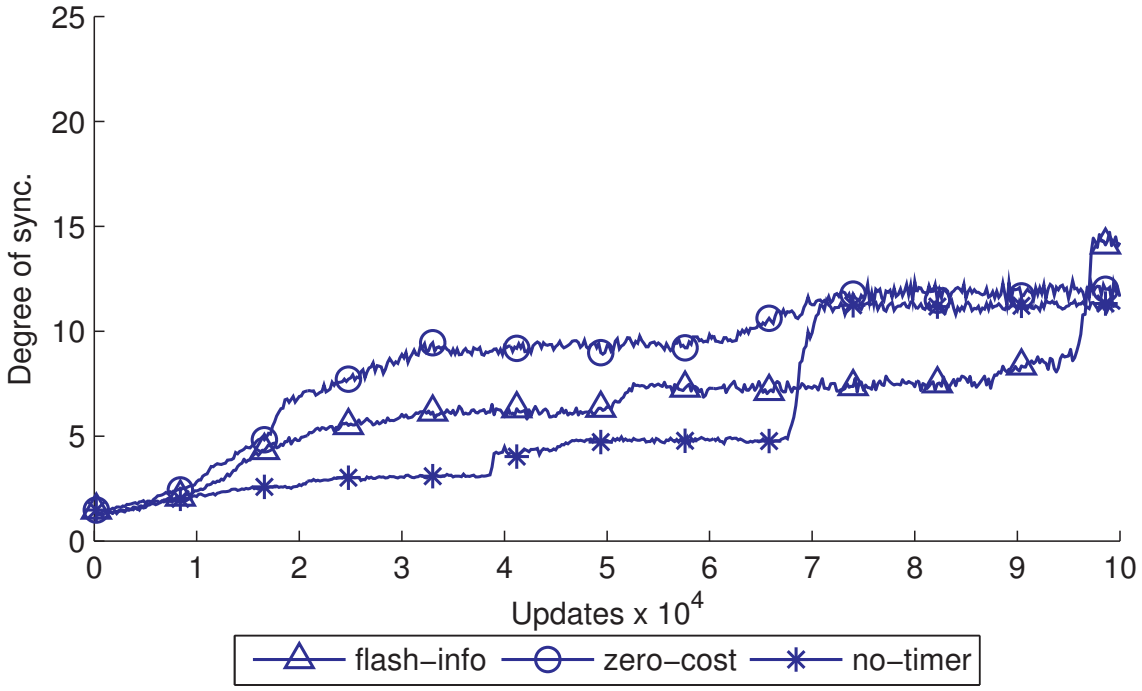


Figure 4.3: Worst-case performance of synchronization trials, showing that the zero-cost treatment is most able to evolve synchronization, and that the availability of timing information improves evolvability.

the execution of the genomes of the organisms in this deme is available at the following URL:

`www.cse.msu.edu/thinktank/firefly`.

4.2.2 Genome analysis

The Avida TestCPU enables the analysis of a single organism in an isolated environment. To analyze the synchronization and desynchronization behavior of an Avida genome, we extended the TestCPU to support the artificial delivery of a flash message to an organism under test. We then traced the organism's execution flow that resulted from receiving a flash at different times, and monitored its response in terms of flash execution.

Here we analyze the dominant genome from the end of trial 1 under the *flash-info* treatment. Trial 1 was one of the highest-performing trials in the synchronization experiment, as

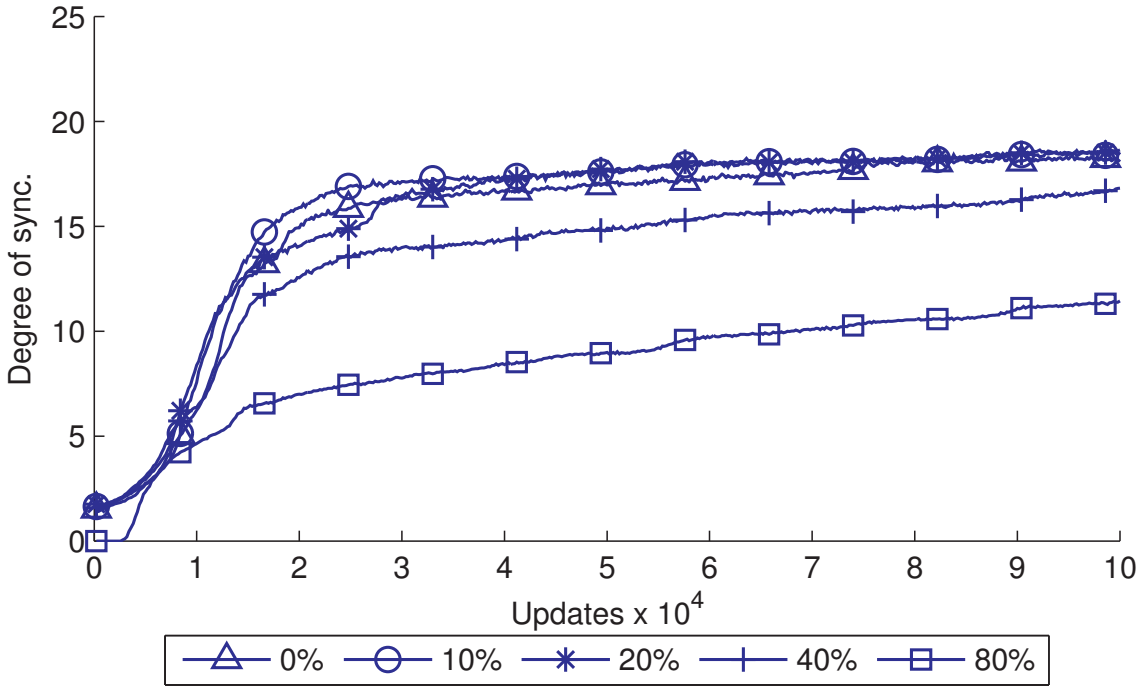


Figure 4.4: Evolution of synchronization over different uniform flash loss rates.

determined by the CompeteDemes fitness function. This treatment used a 20 cycle cost for issuing the flash instruction, and used flash-info, which provided the organism with timing information of the last message received.

Figure 4.5 plots the response of the dominant to receiving a flash at various times during its life. At each point t along the horizontal axis, we initiated a new test of the dominant, and artificially sent a single flash message to the organism once it had executed t virtual CPU cycles. We then monitored the response of the organism from each test for 2000 cycles, and plot a point for each cycle spent executing the flash instruction. For example, in Figure 4.5 at time 100, we see a series of horizontal bands every 110 cycles. This indicates that if the organism receives a single flash after it has executed 100 instructions, it will respond by flashing every 110 cycles. The gaps from time 1 to 19 and near time 50, 110, and 150 are artifacts of the organism executing the hard-reset instruction.

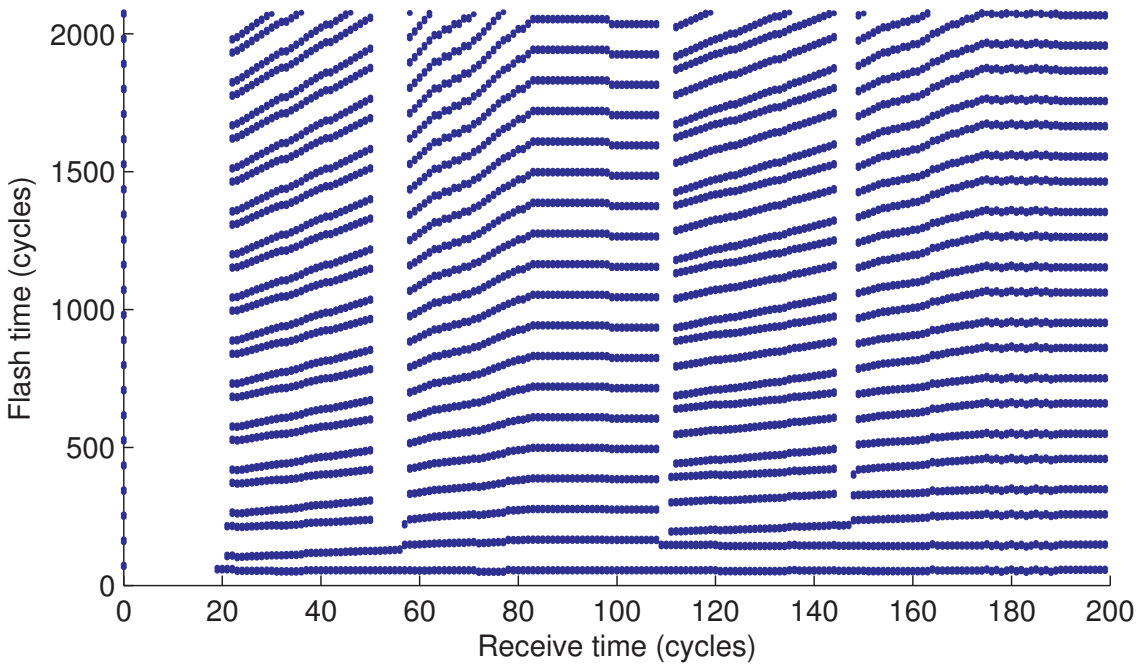


Figure 4.5: Synchronization response of the dominant genome to receiving flash messages at different times.

Figure 4.6 zooms in on the first 180 cycles of the organism's response to receiving flash messages at different times. Here we see evidence of the strategy employed for synchronization. First, the horizontal band extending from time 19 to 200 indicates that, regardless of flash reception, organisms will issue a flash that will complete by their 66th virtual CPU cycle. Region C in Figure 4.6, extending from time 108 to 200, shows that the organism has the same response to receiving a flash during this time, indicating that the organism is not making any phase or frequency adjustments to its flash execution. Region A, on the other hand, shows a quicker execution of flash than region C, indicating that the reception of a message in this region causes a frequency-advance of the receiver. Finally, region B shows a slower response to flash messages than region A, indicating a region of frequency-delay. This combination of steady-state and adaptive frequency operation is similar to the models of biological synchronization from Mirollo, Strogatz, and Ermentrout [48, 47].

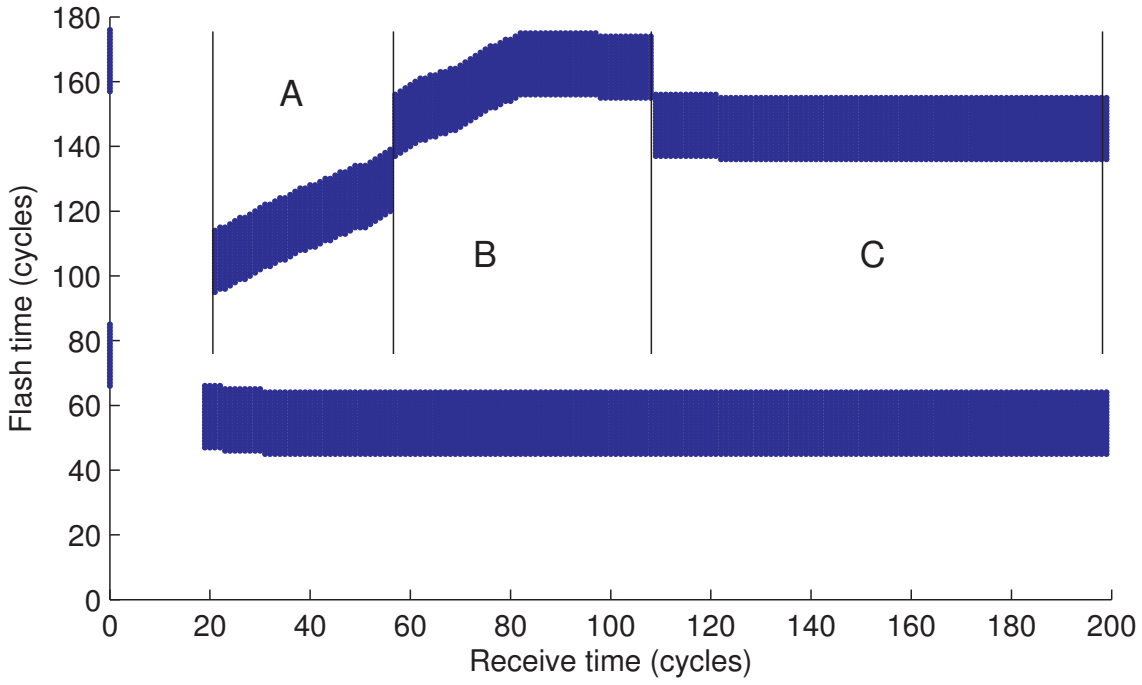


Figure 4.6: Detailed synchronization response of an organism to receiving flash messages at different times. Region C is a steady-state response, while regions A and B are frequency-advance and frequency-delay responses, respectively.

Figure 4.7 depicts the genome responsible for the synchronization behavior shown in Figure 4.2, and highlights the primary instructions. Following birth, execution flow of this genome proceeds nearly linearly for 65 cycles, after which point it begins periodic execution of the flash instruction. However, if a flash is received, its execution flow dramatically changes. Specifically, it will use a combination of hard-reset and jmp-head, an instruction that enables numeric manipulation of the organism's instruction pointer, to either advance or delay the time at which flash will be executed. Interestingly, the organism will use its age as returned by get-cycles and the elapsed time since a message had been received from flash-info to modify the destination for the jmp-head instruction, providing an additional means of frequency adjustment. Finally, we note the large number of instructions within this genome that have no discernible purpose. These instructions are most likely neutral mutations, or

mutations that do not adversely affect fitness. We note, however, that their presence plays a role in determining the native flash period, as they still require virtual CPU cycles to be spent on their execution.

4.3 Evolution of Desynchronization

For this part of the study, our goal is to discover behaviors that bring digital organisms out of phase with respect to each other, similar to round-robin scheduling [49]. As much as possible, the desynchronization experiment mirrors our study of synchronization – Avida configuration and experiment treatment design are, in fact, identical. Only two factors are different: First, we initialized each deme to be in a state of synchronization (that is, all organisms are "born" at the same time and in the same state), and second, we altered the fitness function used. That the opposite temporal behavior can be evolved with only these changes illustrates the flexibility of Avida to evolve such behaviors.

Fitness function. We defined a fitness function that rewards demes whose constituent organisms each execute the flash instruction, and then further rewards demes for decreasing the number of organisms that flash during the same update. Specifically, the fitness function F used for desynchronization was defined as:

$$F = \begin{cases} 1 + U & \text{if } U < S \\ 1 + U + (S - max(C))^2 & \text{if } U \geq S \end{cases} \tag{4.2}$$

where again, U is the number of unique organisms that issued a flash instruction, S is the number of organisms in the deme, and $max(C)$ is the maximum number of flashes during

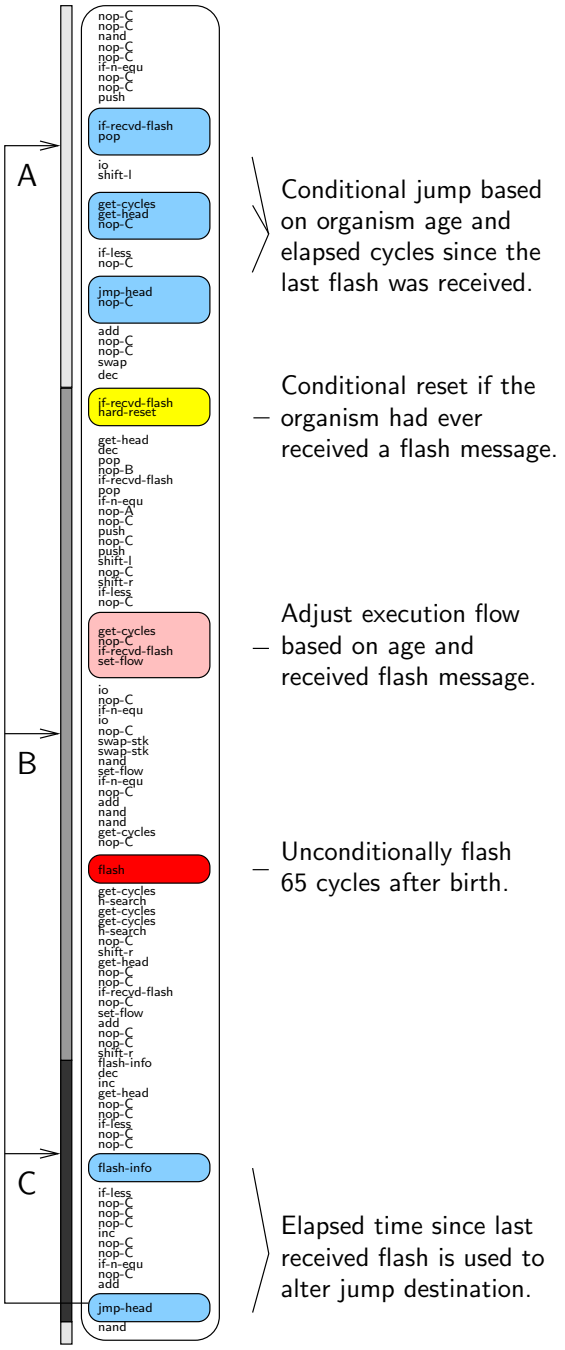


Figure 4.7: Depiction of the dominant genome for synchronization. Labels (A, B, C) refer to destinations of the jmp-head instruction that correspond to frequency-advance, frequency-delay, and steady-state regions from Figure 4.6. Instructions relevant to synchronization are shaded and annotated; all others appear to be neutral mutations.

any single update; both $max(C)$ and U were calculated from the final 50 updates of each CompeteDemes period. This fitness function thus rewards demes whose constituent organisms

all issue the flash instruction, and further rewards for increasing flash desynchronization. Maximum fitness is achieved when all organisms in a deme flash at different times during the final 50 updates of a competition period.

4.3.1 Experimental Results

As with synchronization, we conducted three different desynchronization treatments, *flash-info*, *zero-cost*, and *no-timer*. As before, these treatments used a 20-cycle cost for the flash instruction, a 0-cycle cost for flash, and replaced the flash-info instruction with flash-info-b, respectively.

Figure 4.8 plots the grand mean number of coincident flashes versus updates averaged over each deme in all 30 trials, from each of these treatments. Here we see that over the course of the experiment, the level of desynchronization, measured as worst-case coincident flashes, dropped from 25 (entire deme flashing in unison) to an average of approximately 6.1 after 100,000 updates. (In the best performing trial, this value dropped to approx. 5.0.) We note that the final results of these treatments are not statistically significantly different from each other (Kruskal-Wallis multiple comparison, test for significance performed at the $p < 0.05$ level; standard deviations at final update: $1.10, 1.26, 1.20$), which indicates that desynchronization does not depend on instruction cost or knowledge of the exact timing of message reception.

As with the study of synchronization, we also examined the effect of a flash loss rate on the evolution of desynchronization. We again varied loss rate from 0% to 80%, and used a unit-cost flash instruction. Figure 4.9 plots the number of coincident flashes versus time averaged over each deme in 10 trials, iterated over different flash loss rates from 0%-80% loss under

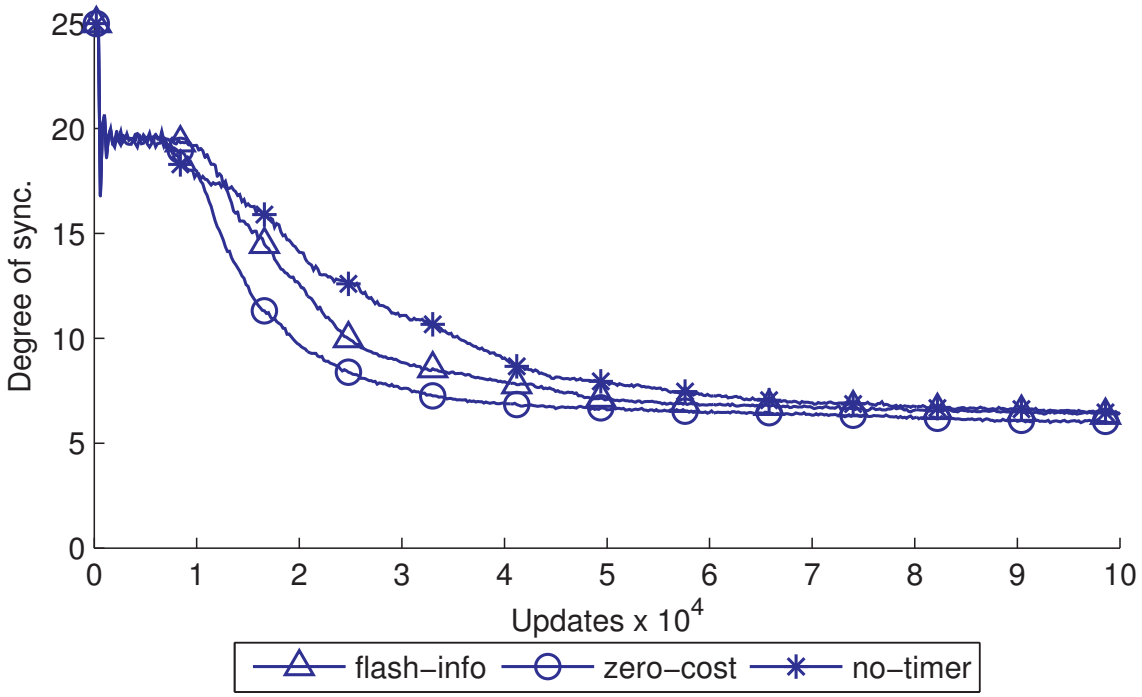


Figure 4.8: Evolution of desynchronization using a fitness function that rewards for desynchronization in execution of the flash instruction, under three different treatments.

the desynchronization treatment. Interestingly, this figure shows that desynchronization evolves more quickly under the 80% loss rate than under the 0% loss rate, which shows that evolution can take advantage of characteristics of the environment (in this case, random flash loss) to solve a particular problem. Here, not only does the random loss of a flash decrease the likelihood of coincident flashes, but these losses also serve as a source of randomness, which assists individual organisms in spreading their flashes throughout phase-space. No statistically significant differences of the final results were found between these treatments (Kruskal-Wallis multiple comparison, test for significance performed at the $p < 0.05$ level).

Figure 4.10 is a detail of the behavior of a single deme containing 25 copies of the dominant genome from the end of one of the best-performing Avida flash-info trials. Each point again represents the execution of the flash instruction by a particular organism. Here, however, we see that all organisms initially emit a single flash in unison near update 20, and thereafter

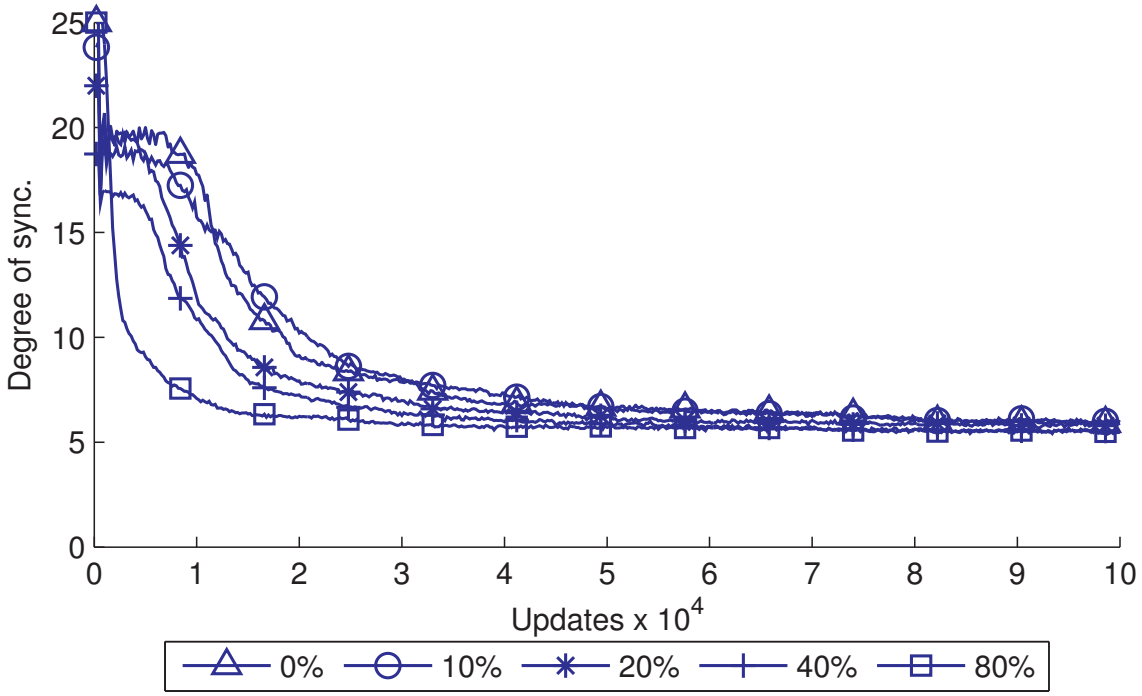


Figure 4.9: Evolution of desynchronization over different uniform flash loss rates.

progressively increase the level of desynchronization in the deme. We note that this evolved behavior is not perfectly desynchronized; there are 4 coincident flashes near update 175.

4.3.2 Genome Analysis

We analyzed the dominant genome from the end of trial 14 of the flash-info treatment. Trial 14 was one of the highest-performing trials in the desynchronization experiment. As with synchronization, this treatment used a 20 cycle cost for issuing the flash instruction, and used flash-info to provide the organism with timing information of the last message received.

Figure 4.11 plots the response of the dominant to receiving a flash at various times during its life, similar to Figure 4.5. At each point t along the horizontal axis, we initiated a new test of the dominant, and artificially sent a single flash message to the organism once it had executed t virtual CPU cycles. We then monitored the response of the organism from each

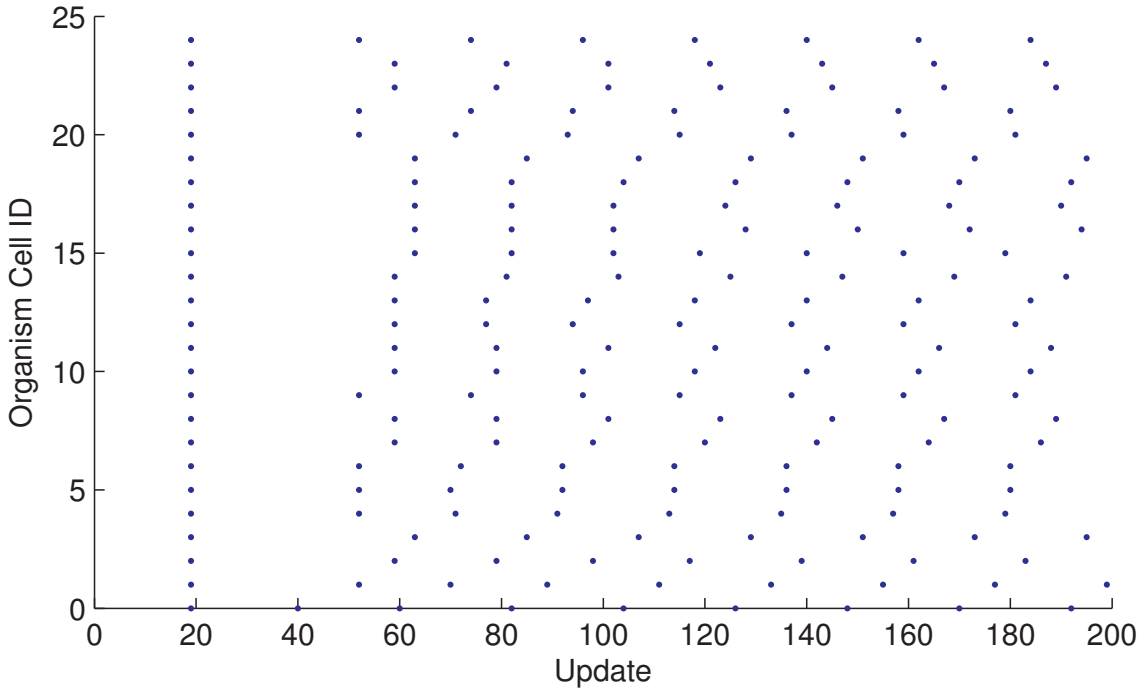


Figure 4.10: Detail of evolved desynchronization behavior. Organisms initially flash in unison, but gradually increase the level of desynchronization present.

test for 2000 cycles, and plot a point for each cycle spent executing the flash instruction. As before, the gap from time 13 to 19 is an artifact of the organism executing the hard-reset instruction.

Figure 4.12 zooms in on the first 200 cycles of the organism's response to receiving flash messages at different times, enabling us to characterize the strategy employed for desynchronization. First, the horizontal band extending from time 57 to 199 indicates that, regardless of flash reception, organisms will issue a flash that will complete by their 97th virtual CPU cycle. Regions A and C exhibit the same frequency-advance behavior as with the synchronization dominant, while regions B and D exhibit steady-state behavior, although at slightly different frequencies.

Figure 4.13 depicts the instructions present in the dominant genome, and shows the primary instructions that are responsible for the behavior in Figure 4.10. There is a single

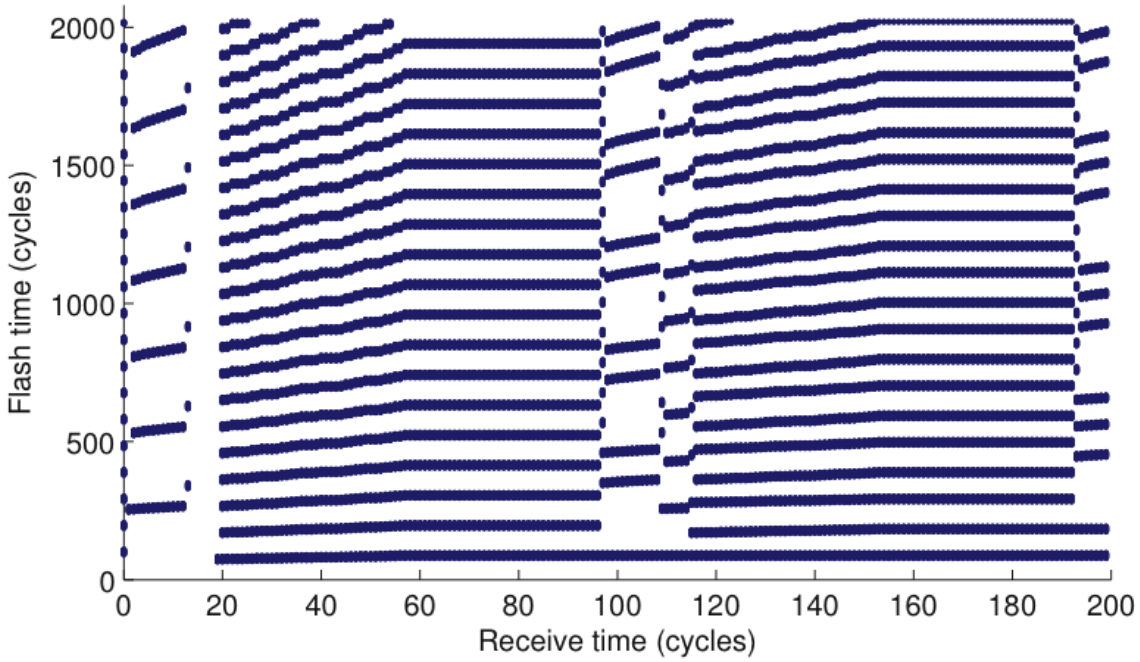


Figure 4.11: Desynchronization response of the dominant genome to receiving flash messages at different times.

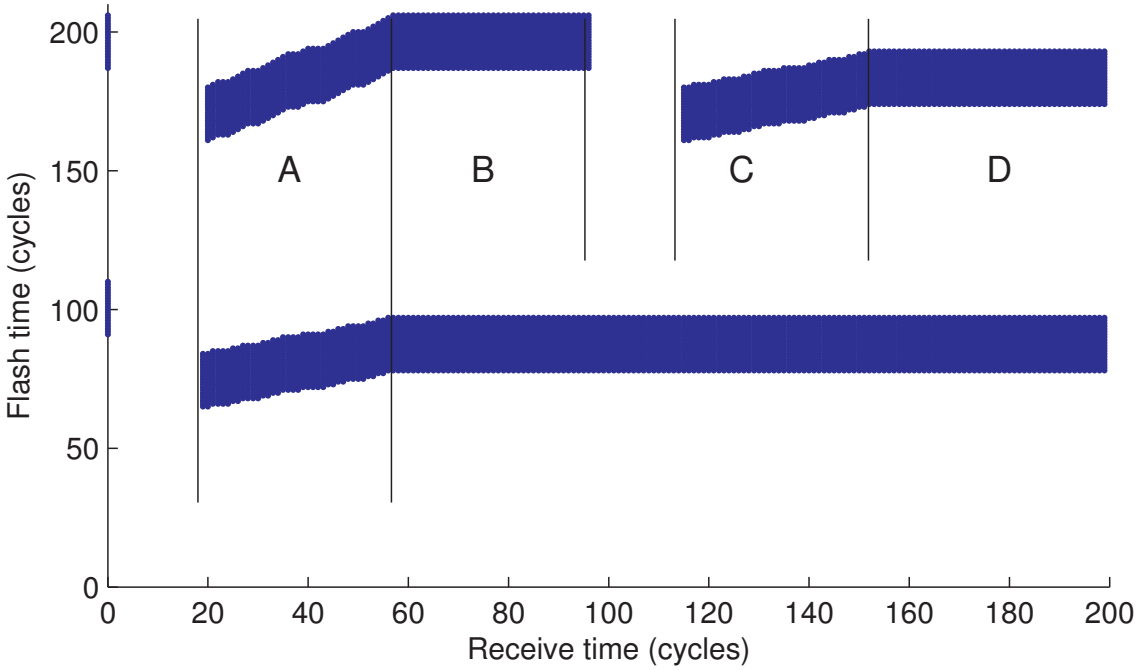


Figure 4.12: Detailed desynchronization response of an organism to receiving flash messages at different times. Regions A and C are frequency-advance responses, while regions B and D are steady-state responses.

flash instruction present in this genome. Upon birth, and when it does not receive a flash message, this organism executes the first 91 instructions approximately in order, completing its own flash at cycle 110. As with the synchronization dominant, if this organism receives a flash at any point during its lifetime, its execution flow changes. Here, however, rather than using the age of the organism and the elapsed time since the last flash message was received, this organism modifies the position of its write-head (a pointer into its genome, similar to a stack pointer) in response to received flash messages to determine the next point in time at which to execute the flash instruction. Similar to the engineered algorithms in [49], this genome attempts to iteratively adjust the organism's phase and frequency until it is desynchronized. Here, however, the evolved strategy is dependent on local information only – it neither requires access to its neighbors in phase-space, nor are flashes sent globally.

4.4 Conclusion

In this chapter, we have shown that digital evolution can produce temporal behaviors for synchronization and desynchronization when provided with primitives based on biological fireflies and purely local information. The evolved solutions utilized an adaptive frequency strategy, similar to models of firefly behavior, that altered their control flow based on a combination of sensed information and the organism's age. We have also shown that these relatively complex cooperative behaviors can be evolved via simple fitness functions, that both synchronization and desynchronization can be evolved in Avida by a straightforward change in fitness functions, and that the evolution of these behaviors is robust to message loss.

We now have strong evidence that digital evolution, specifically Avida, is capable of

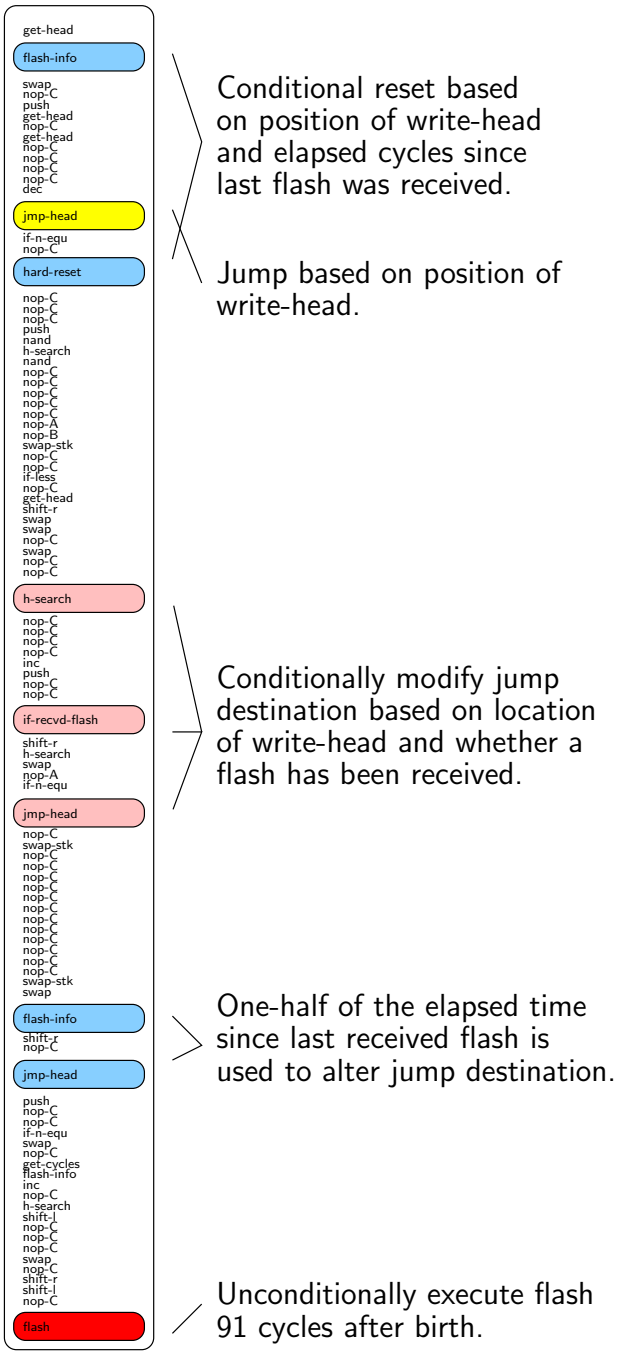


Figure 4.13: Depiction of the dominant genome for desynchronization. Instructions relevant to desynchronization are shaded and annotated; all others appear to be neutral mutations.

evolving two important classes of distributed behavior. These results demonstrate that digital evolution could be an important tool for helping us to engineer distributed computing systems. However, we have yet to investigate whether digital evolution is capable of discov-

ering structural behavior, a subject which we explore in the next chapter.

Chapter 5

Evolving Structural Behavior: Network Topology

An underlying feature of all communication is that it occurs within a network of some kind. The structure of that network, whether physical or logical, is a critical part of any distributed behavior. Of course, the natural world is filled with examples of communities of organisms that cooperate to build complex structures. For example, biofilms are complex extracellular structures that are formed by nearly all species of microorganism [5], while the eusocial *Hymenoptera* (sawflies, wasps, bees, and ants) build complex nests that house many thousands of individuals [6]. Some of the more striking examples of animal-built structures are the mounds built by the African termite subfamily Macrotermitinae. In this case, the mounds not only house the colony, but also provide thermoregulation and ventilation for the fungus that these termites farm [140].

Distributed computing systems, in contrast, are highly effective at building communication networks. However, the task of establishing and maintaining a network topology can

be complicated by physical terrain, node movement, and dynamic environmental conditions that affect signal propagation [141]. Moreover, an effective strategy must maintain network connectivity while conserving energy. While a wide variety of approaches to topology control have been designed for wireless ad hoc networks [141], some of the most promising algorithms proposed recently are inspired by biology [24, 142]. For example, Wakamiya et al. recently described a reaction-diffusion model where the equations for activation and inhibition are based on patterns drawn from angelfish [142, 143].

Our studies: In this chapter, we describe our results in using digital evolution to discover cooperative behaviors for network construction. To determine if digital organisms in Avida are in fact sensitive to their underlying network topology, we start with a study in the evolution of a simple data-distribution behavior on various fixed-topology networks (the data distribution problem is described in Section 5.1). Next, we present the results of a study in the evolution of network construction, where we evolve digital organisms to construct networks that have various properties, but without a higher-level task. That is, we reward for the structure of the network itself, as opposed to *using* the network. Finally, we conclude with a study in which we evolve digital organisms that not only construct networks, but also use them for the data-distribution task. Taken together, these results suggest a new class of distributed algorithm, where agents construct a network specific to the task at hand.

5.1 Methods

To evolve network construction, we extended Avida with a small set of instructions that enabled the creation of a *logical network* among organisms in an Avida population. In

contrast to the cell connections and environment topology described in Section 2.3.1, logical networks are unrelated to the underlying physical topology, and are instead more closely related to overlay networks, as described in Section 2.1. The key difference between prior studies and those described here is that we are now adjusting the network organisms use for communication independently of their location in the underlying spatial environment. As with previous studies, here we also used digital germlines and deme competition.

Instructions. To enable communication and the construction of logical networks, we implemented 7 new virtual CPU instructions and leveraged both the communication- and opinion-related instructions described in Sections 2.3.2 and 3.4. The new instructions are summarized in Table 5.1. These instructions were jointly inspired by existing Avida instructions that enabled interaction among neighboring organisms, as well as by network sockets, where a connected socket is analogous to the construction of a logical network link. The different variants of the create-link-$*$ instruction are used to construct logical network links among organisms. For example, the create-link-facing instruction creates a logical link between the caller and the organism that is currently faced, while create-link-xy creates a link between the caller and a cell designated by (x, y) coordinates. We note that the links created by these instructions are bi-directional; the link can be used by organisms on both ends of the link. The network-unicast instruction sends a single message to a neighboring organism; the network-rotate and network-select instructions are used to alter the connection that is used by network-unicast. Finally, the network-bcast1 instruction broadcasts a message from the caller to all organisms to which it is connected.

Table 5.1: Relevant instructions for this study. All instructions are equally likely to be selected as targets for mutation.

Instruction	Description
create-link-facing	Create a network link between the caller and the organism currently faced.
create-link-xy	Create a network link between the caller and the cell in the deme specified by the (x, y) coordinates in the BX and CX registers.
create-link-index	Create a network link between the caller and the cell with the index specified in BX.
network-rotate	Rotate the currently-active network link by the value specified in BX.
network-select	Activate the network link specified by the value in BX.
network-unicast	Send a message with data and label fields set to (BX, CX) via the currently-active network link.
network-bcast1	Broadcast a message to all organisms the caller is connected to.

Table 5.2: 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	Clique
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, probabilistic scheduler
Ancestor	100 nop-C instructions
CompeteDemes	Compete all demes using (1,2) tournament selection (periodic, every 400 updates).

Configuration. In the majority of the experiments described in this chapter, we configured Avida to use the CompeteDemes process, outlined in Section 2.3.3, with specific configuration values summarized in Table 5.2. We used 400 demes, each comprising 25 digital organisms, with each deme configured to use a germline to provide homogeneity within demes. The default values were used for all mutation rates, and the time slice was again set to 5 instructions per update.

In order to control the initial conditions for each deme, we again configured Avida to

disallow self-replication of individual organisms. Each time a deme replicated as a result of the CompeteDemes process, the offspring deme was filled with 25 copies of the latest (possibly mutated) genome from the germline. We note that deme replication does *not* include the logical network; offspring demes are initialized with an empty network.

5.1.1 Data Distribution on Fixed and Constructed Networks

For many of the experiments described in this chapter, we focus on the interplay between the structure of the communication network and the ability of digital organisms to solve a simple data distribution task. An example of this data distribution task on a network of fixed structure is depicted in Figure 5.1. Figure 5.1(a) depicts 25 organisms "living" in a deme (for clarity, here we depict a grid-based deme and network; both the underlying environmental topology and communication network vary in the following studies). A single cell in each deme is designated as the initial source of data (cell data is set to a random 32-bit number) for that deme. The individual living in that cell, shaded in this figure, is responsible for initiating data distribution by collecting the data in its cell and sending messages containing that data to its neighbors. In Figure 5.1(b), we see that neighbors of the source have received the data, again indicated by shading, and have in turn forwarded it to other individuals. The number of organisms that receive this data will be used as the basis for fitness functions that will be presented later. Finally, in Figure 5.1(c), we see that the data has been distributed to all individuals within the deme. At this point, the data distribution task is complete.

An illustration of joint data distribution and network construction is depicted in Figure 5.2. Figure 5.2(a) depicts the initial conditions of a deme at the time of its birth. Here we see 25 digital organisms, one of which occupies a cell that has again been initialized with

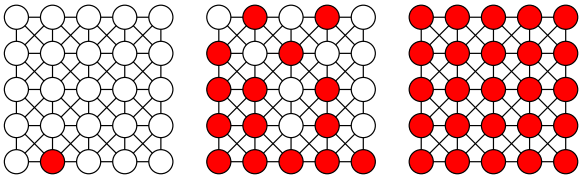


Figure 5.1: Depiction of data distribution in a 5×5 grid. In (a), data is present at a single cell only, shown by the shaded circle; (b) depicts organisms in the midst of distributing data; finally, (c) shows complete distribution of data.

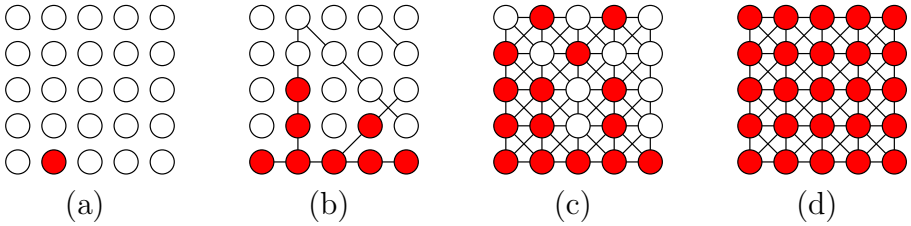


Figure 5.2: Depiction of data distribution in a deme of 25 organisms. In (a), data is present at a single cell only, shown by the shaded circle, and no network links are present; (b) depicts organisms in the midst of network construction and data distribution; (c) shows the completion of network construction, as all organisms are connected; finally, (d) shows complete distribution of data.

random cell data. At this point, no logical network links are present. In Figure 5.2(b), we see that a number of individuals in the deme have begun to construct a network, and that neighbors of the source have received the data. In Figure 5.2(c), we see that all organisms are connected to the same logical network (that is, the network is *connected*), though not all have received the data. Finally, in Figure 5.2(c), we see that the data has been distributed to all individuals within the deme. At this point, both the network construction and the data distribution tasks are complete.

Whether the network topology is fixed or constructed by digital organisms, solving the data distribution task depends on two factors: First, the data must be distributed among all organisms in the deme, and second, organisms must indicate that they have, in fact, received this data. For this data distribution task, we require that organisms acknowledge

receipt of the data by setting their opinion register (using the set-opinion instruction defined in Section 3.4) to the value of the data they have received. The number of organisms in the deme that have set their opinion register correctly are used as the basis of fitness functions that follow.

5.2 Data Distribution on Fixed Topologies

In this section, we describe our results in using Avida to investigate the effects of various communication network topologies and pathologies on the evolution of organisms that solve the data distribution task on fixed-topology networks. Each of the experiments described here focuses on a single aspect of this problem, and each was designed to provide insight into how different network structures and environmental factors affect the evolution of this cooperative behavior. In this study, we progress from relatively simple single-criteria problems to more complex multi-objective optimization problems.

5.2.1 Basic Distribution of Data

Our initial experiment examined the effect of different underlying topologies on the evolution of data distribution, without regard to efficiency of messaging. Specifically, we varied the topology of each deme's communication network between a torus, clique, scale-free, and a random connected network (mean degree approximately 3.25), while employing a fitness function based solely on the number of individuals that received the data. We defined the fitness function F for each deme as:

$$F = (1 + U)^2 \tag{5.1}$$

where U is the number of organisms within the deme that indicate they have received data. Organisms indicate that they have received the data by executing the set-opinion instruction with the received data passed as a parameter.

Figure 5.3 plots the grand mean fitness of each deme for each of the different underlying network topologies across 30 trials. In this figure, a black horizontal bar is used to indicate the fitness of a deme that has successfully completed the data distribution task, in this case a fitness of 676 ($F = (1+25)^2 = 676$). While the grand mean fitness of each treatment is below this level, we note that at least one deme in each treatment successfully distributed data to all organisms by update 20,000. Although roughly the same number of organisms received the data in each of the four different topologies, suggesting that there is little difference in performance among the different topologies, we found the fitnesses of all treatment pairs to be significantly different from each other (Wilcoxon rank sum test, $p < 2 \times 10^{-6}$), with the exception of the random and scale-free treatments, which performed similarly to each other. These results illustrate that organisms are able to evolve data distribution across networks of varying connectivity.

In this experiment, the dominant evolved genomes that solved the data distribution problem employed a fairly succinct looping behavior. Figure 5.4 depicts the Avida instructions for one such loop, drawn from a single trial of the random network treatment. Here, the entire loop is 11 instructions in length, approximately 10% of the genome; the remainder of the genome was "junk DNA," remnants of earlier approaches to data distribution. As shown in Figure 5.4, organisms with this genome would either send messages that contained data sensed at their location within the deme, or forward messages that they had previously received. Additionally, each sent message was followed by a rotation so that the next message

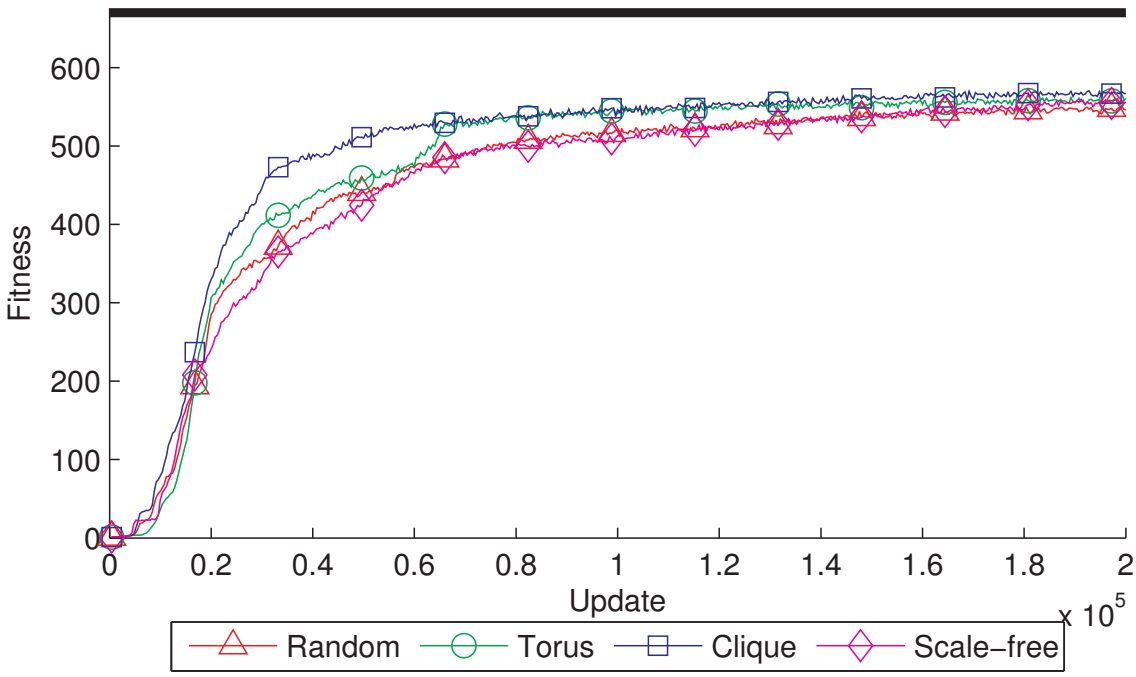


Figure 5.3: Fitness for random, toroidal, clique, and scale-free topologies when rewarding only for data distribution.

would be sent to a different neighbor.

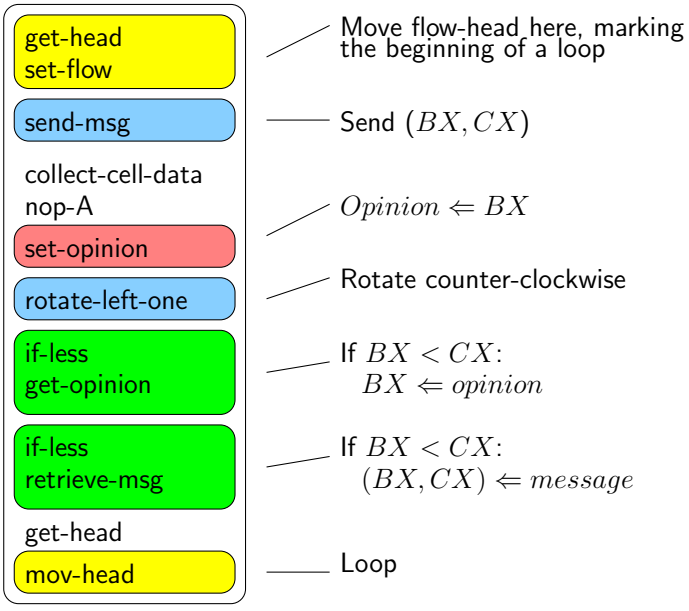


Figure 5.4: Representative genome fragment for solving the data distribution problem, drawn from a single trial using a random network topology. Prior to this loop, this genome contains instructions to test its location for the presence of data, setting its opinion if data is found.

5.2.2 Efficiency on Differing Topologies

In the next experiment we examined the effect of different underlying topologies on the evolution of *efficient* data distribution. Specifically, we varied the underlying topology between torus, clique, scale-free and random connected network, while employing a fitness function based on both the number of individuals that received data as well as the number of messages that were used. We defined the fitness function F for each deme as:

$$F = \begin{cases} (1+U)^2 & \text{if } U < S \\ (1+U+(S*1000)/C)^2 & \text{if } U = S \end{cases} \tag{5.2}$$

where U is the number of organisms within the deme that received and expressed the data, S is the number of organisms in the deme (always 25 in this study), and C is the total number of messages sent by organisms within the deme. We note that S is scaled by 1,000 to reward demes for completing the data distribution task even at the cost of a large number of messages (C).

Figure 5.5 plots the grand mean fitness of each deme for these treatments across 30 trials. As before, a black horizontal bar is used to indicate the fitness at which a deme successfully completed data distribution (note log_{10} scale); any fitness improvements above this level are the result of efficiency gains. Here we see a difference in performance among the different network topologies, with the clique-based treatment reaching a fitness nearly 100 times that of the other treatments. Indeed, grand mean fitnesses of all pairs of treatments are significantly different ($p < 1 \times 10^{-6}$), and as a result we conclude that the underlying topology does play a significant role in the evolution of data distribution when efficiency is a concern.

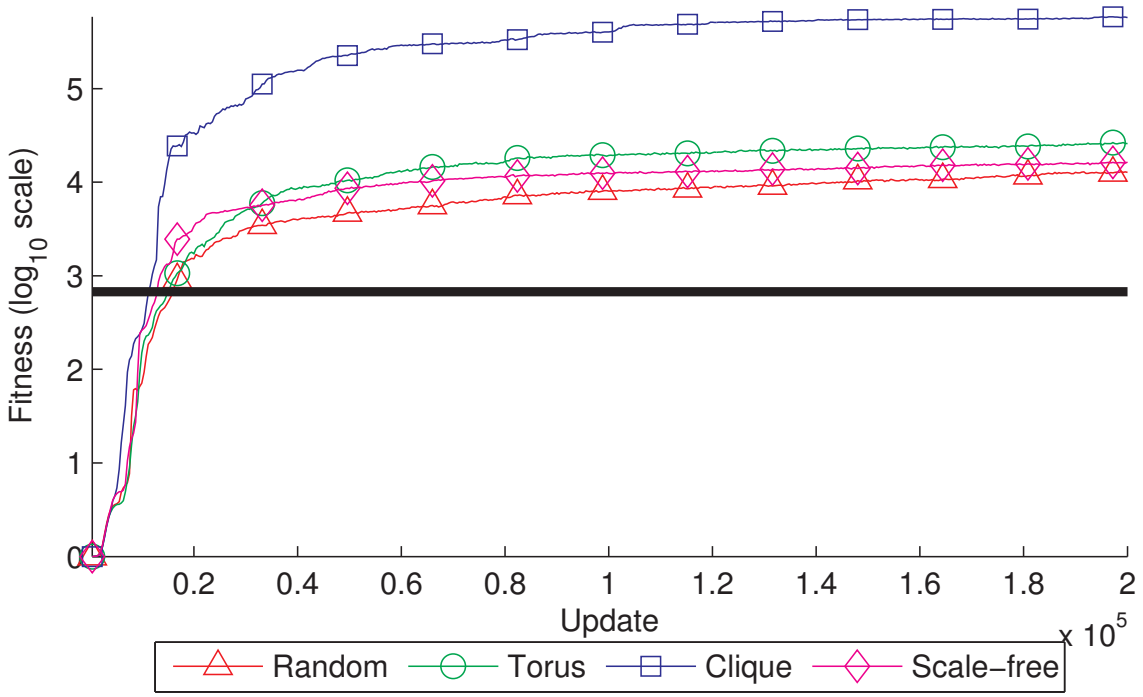


Figure 5.5: Fitness for random, toroidal, clique, and scale-free topologies when rewarding for efficient use of messages.

In this experiment, the highest-performing strategy across all treatments evolved on a clique network, where only the organism at the location containing data sent messages; all other organisms remained silent, but actively listened for messages, setting their opinion once a message was received. Interestingly, the highest-performing deme on a torus network evolved a strategy that depended upon both the contents of messages, as well as the frequency with which messages were received. Psuedo-code for this strategy is shown in Algorithm 5.1. In effect, this strategy uses messages for *behavior suppression*, where receiving a message after an individual has set its opinion results in the suppression of future message sending.

Algorithm 5.1 Representative evolved algorithm for data distribution in a torus when rewarding for efficiency.

```
Require: opinion is null; location is null for all but a single individual within each deme.
  if location is not null then
    opinion ⇐ location
  end if
  loop
    if message is received AND opinion is null then
      opinion ⇐ message
    end if
    if message is not received AND opinion is not null then
      send opinion
    end if
    rotate
  end loop
```

5.2.3 Effects of Message Cost

We next examined the effect of different costs for sending messages. For this and remaining studies, we use only the random connected network topology and employ the fitness function described in Equation 5.2. Here, however, we also varied the cost of sending messages from one organism to another while still rewarding for efficiently distributing data to all individuals. Specifically, we varied the cost of sending a message from 1 to 100 cycles per instruction, testing costs $\{1, 20, 40, 60, 80, 100\}$. Any message sending cost larger than 1 implies that sending a message is proportionally more expensive relative to other instructions, and delays the transmission of data through the network. Thus, a higher cost for sending a message is analogous to limiting the available bandwidth in a network.

Figure 5.6 plots the grand mean fitness of each deme for these treatments across 30 trials. As before, a black horizontal bar is used to indicate the fitness at which a deme successfully completed data distribution. These results illustrate that organisms are capable of evolving efficient behaviors for distributing data, regardless of message cost. In particular, as groups

of organisms were placed in networks with higher message sending costs, they were able to evolve more efficient behaviors for data distribution. Although the grand mean fitness of most pairs of treatments were significantly different from each other ($p < 1 \times 10^{-6}$), treatments with a cost of 20, 40, and 60 were similar to each other.

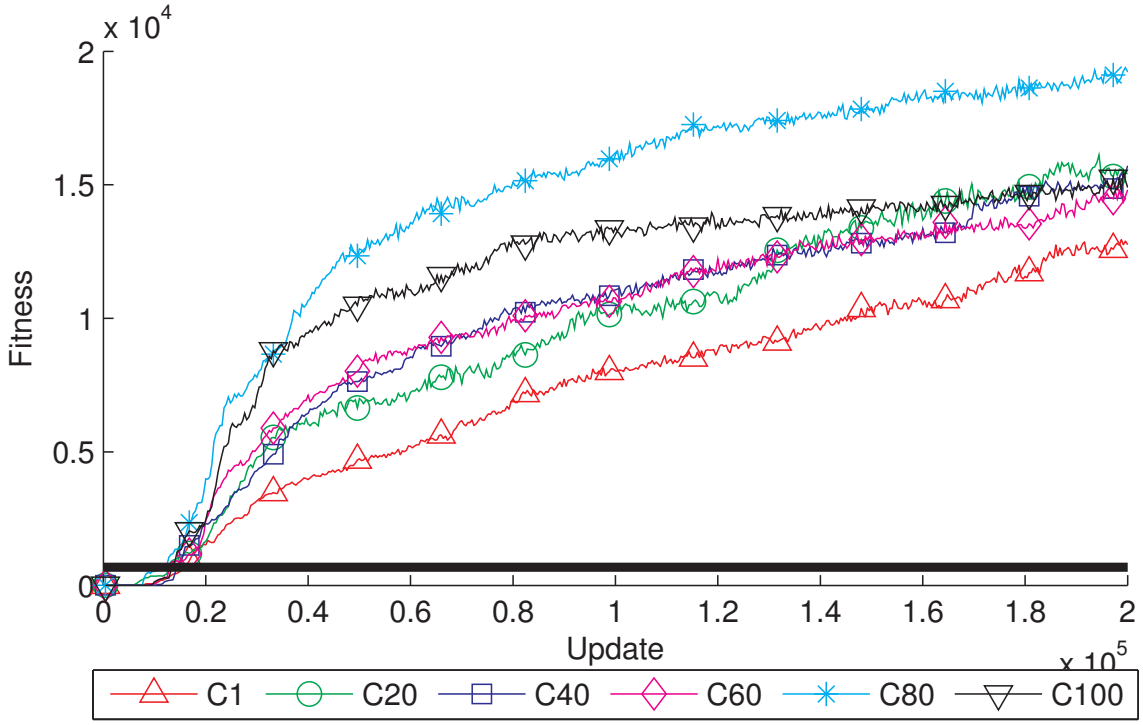


Figure 5.6: Fitness for a range of different message costs. The addition of a cost to send messages improves efficiency.

A side effect of increasing the cost of sending a message is that organisms have fewer opportunities to send messages, thus automatically providing a fitness advantage. Indeed, the evolved genomes verified this observation, as the strategies employed were similar to those described in Section 5.2.1, where efficiency was not rewarded. The treatment with a cost of 80, however, evolved a novel solution to the data distribution problem, where two interlocking loops were used to control organism behavior; psuedo-code is shown in Algorithm 5.2. In this case, the first loop, LOOP-A, is executed until the organism sets

its opinion to either data contained at its location or the contents of a message, whichever contains a non-zero number. Once its opinion is set, however, the second loop, LOOP-B is executed, which repeatedly sends a message containing the organism's opinion, and rotates the organism to face a new neighbor.

Algorithm 5.2 Representative evolved algorithm for data distribution in a random network when rewarding for efficiency and using a message cost of 80.

Require: *opinion* is null; *location* is null for all but a single individual within each deme.

```
LOOP-A:
opinion ⇐ location OR message {null treated as zero.}
LOOP-B:
if opinion is null then
  goto LOOP-A
end if
send opinion
rotate
goto LOOP-B
```

5.2.4 Effects of Message Loss

We next turned to environments that included some degree of message loss. Specifically, we varied the probability (from 0 to 0.8 in intervals of 0.2) that a message would be lost. Here, each dropped message counts as a message that is successfully sent but never received at its destination. Given that we use the deme fitness function defined in Equation 5.2, dropped messages negatively affect fitness in two different ways: they both hinder data distribution and reduce the reward for efficient use of messages.

Figure 5.7 plots the grand mean fitness of each deme for these treatments across 30 trials. This plot illustrates that even moderate message loss rates result in significant drops in fitness.

Somewhat surprisingly, we note that although fitness varies with message loss, message

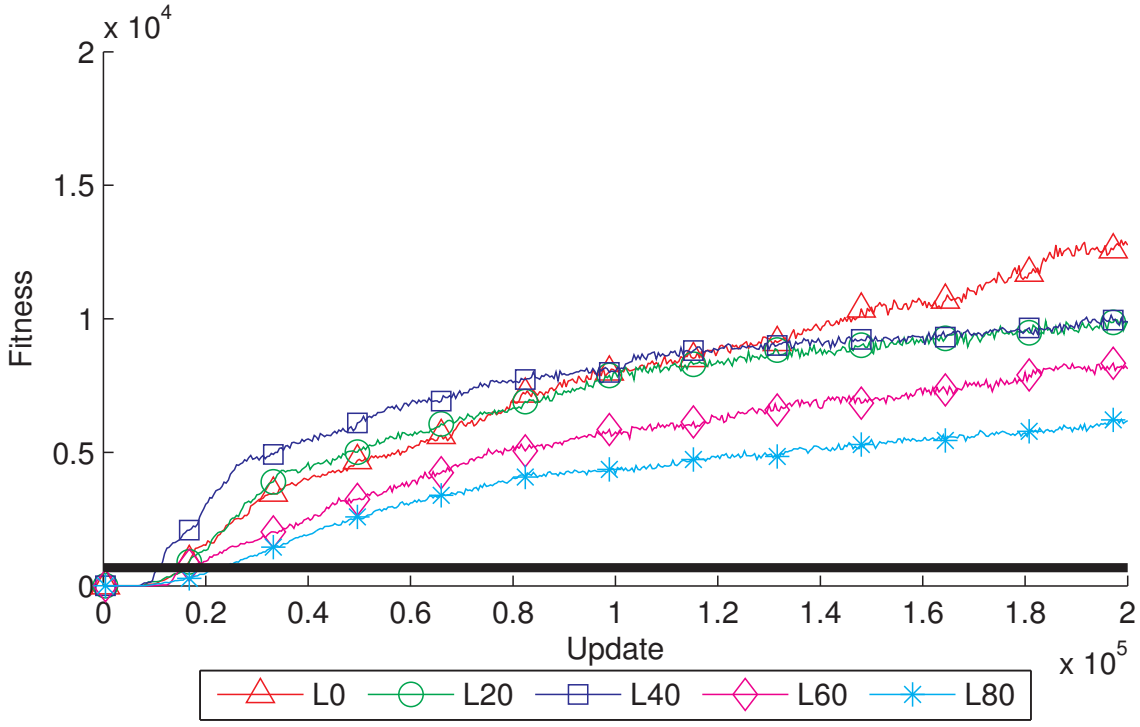


Figure 5.7: Fitness for a range of message loss rates.

counts approach a common equilibrium across each treatment, as shown in Figure 5.8. As in earlier experiments, the evolved behaviors followed a wait-and-forward strategy similar to that described in Algorithm 5.1. However, the number of instructions present in these loops varied with loss rate, with higher loss rates generally having shorter loops, thus increasing the number of attempts to send messages.

5.2.5 Effects of Node Churn

Our next experiment focused on the effect of node churn, that is, node death and replacement, on the evolution of efficient data distribution. As in earlier experiments, the fitness function used here rewarded for data distribution and efficiency, however, here we also probabilistically replaced organisms within each deme. When an organism is replaced, it is killed and a new organism, with its genome drawn from the germline, is instantiated in its place. We tested

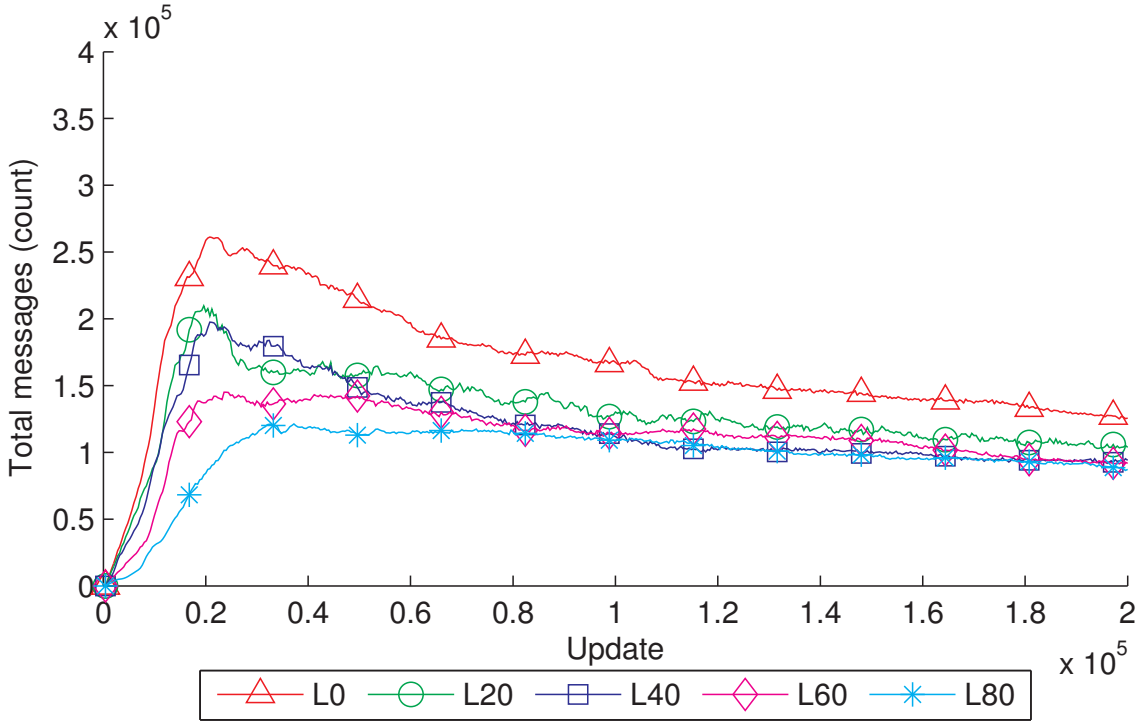


Figure 5.8: Messages sent across a range of message loss rates.

multiple treatments for different values of this replacement probability ranging from 0 to 1, in increments of 0.2. The frequency of checking for node replacement and associated probabilities were selected to provide, on average, 25 replacements per deme at a probability of 1.0. We note that when a node is replaced, the neighboring organisms must re-send the data to that location in the network, and that until they do so, any reward for efficiency is unavailable (as that individual cannot set its opinion without first receiving data).

Figure 5.9 plots the grand mean fitness of each deme for these treatments across 30 trials. Here, higher probabilities of node churn resulted in lower fitness values, indicating that replacement has a significant impact on the evolution of data distribution, and all pairs of grand mean fitnesses are significantly different from each other ($p < 1 \times 10^{-6}$). We note that in each case, the fitness of all treatments is still gradually increasing at the end of the experiment. This, in combination with the variance in fitness among treatments

with a non-zero churn, indicate continued adaptation to this environment. As with the previous experiment, the strategies that evolved to solve this problem are similar to that in Algorithm 5.1.

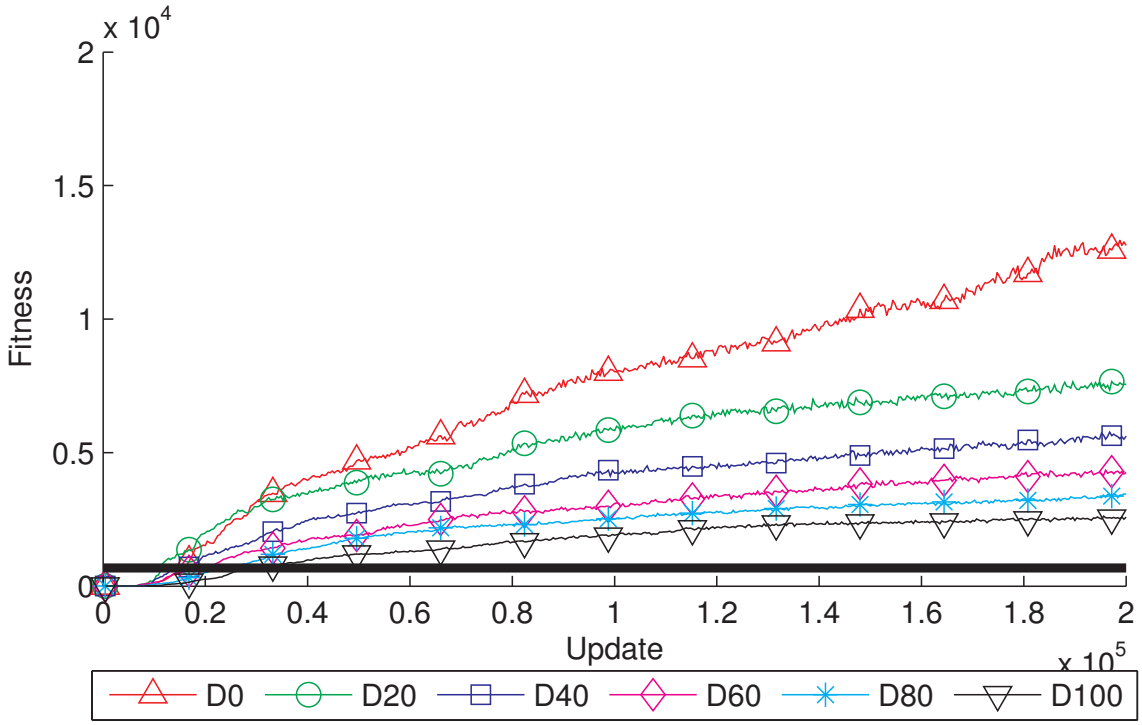


Figure 5.9: Fitness for a range of probabilities of node churn.

Interestingly, when comparing Figure 5.9 to Figure 5.7, we see that even modest levels of node churn have a much greater detrimental effect on fitness than do increased loss rates. This result suggests that designing systems that are resilient to churn would be more beneficial than reducing loss rate.

5.2.6 Data Distribution in the Presence of Multiple Pathologies

In the final experiment we examined the combined effect of message loss, message cost, and node churn at different levels, while still rewarding for efficiency. The tested configurations included: $\{loss = 0.1x,\ churn = 0.1x,\ cost = max(1, 10x)\}$ for $x \in \{0, 1, 2, 3\}$. For example,

we tested $\{loss = 0.1,\ churn = 0.1,\ cost = 10\}$. Figure 5.10(a) plots the grand mean fitness of each deme under three different levels of loss, cost, and churn versus the control, each across 30 trials. Here we see that each treatment that includes positive rates of loss, cost, and churn have significantly lower fitness than the $\{0,\ 0,\ 0\}$ treatment. Figure 5.10(b) plots the grand mean message counts across these same treatments. Interestingly, the lower values of cost, loss, and churn reach a higher initial peak in message sending, with the 30-30-30 treatment never peaking, instead reaching an equilibrium early on. Also, we note that all treatments converged on approximately the same level of messaging activity by the end of the experiment, and that the trends visible in earlier experiments are also visible here, suggesting independence. The early equilibrium of message cost, as well as the slight peak displacement of message loss are both visible in this plot.

Evolved solutions to this experiment were similar to that described in Algorithm 5.2. Of all the different experiments conducted here, we note that structures similar to Algorithm 5.2 have appeared only in experiments containing cost, and that the strategy itself contains two different behaviors, each with a distinct period. Although it is possible that the evolution of this structure is an artifact of Avida, an alternative explanation is that agents within communication-based systems, especially those that include cost, operate most effectively at multiple different frequencies. Moreover, this structure encodes multiple states, where the behavior of an individual in terms of loop frequency and messaging behavior, depends on its location in the environment and whether the individual has received a message.

The experiments presented in this section have shown that Avida is capable of evolving organisms whose data-distribution behavior depends on the structure of the underlying network and the presence of various network pathologies in the environment. We next in-

vestigate the evolution of behaviors where organisms construct the network itself.

5.3 Construction of Connected Networks

In this section, we focus on the evolution of digital organisms that construct connected networks, where there is at least one path between every pair of individuals in a deme. We first show that this task requires homogeneity within demes. We then explore simple selective pressures that can alter characteristics of these networks, such as reducing link usage. Next, we investigate whether more complex selective pressures can be used to build networks that balance multiple criteria, for example, networks that have both a low link count and a small diameter. Finally, we show that this approach can construct networks that meet complex criteria, such as a specific clustering coefficient or characteristic path length.

Methods. In contrast to many other studies presented in this dissertation, this series of experiments uses the Avida ReplicateDemes framework (instead of CompeteDemes). The reason for this is straightforward: Here, we desired organisms to construct a connected network as quickly as possible. By using the ReplicateDemes framework, we were able to replicate demes as soon as they built a connected network, thus establishing a "race to connectedness" among demes. This model is related to Traulsen's model of multilevel selection [60], where it is hypothesized that groups first grow to a sufficient size prior to division. To bootstrap the replication process, we configured the ReplicateDemes framework to replicate a deme as soon as its inhabitants had constructed a connected network or 100 updates had passed, whichever was first.

We do still use a fitness function in these studies to provide a stronger selective advantage

to demes that construct networks with desirable properties. Here, however, the fitness functions are used to augment the merit of all organisms in the group (c.f. AVIDA tasks, which apply only to a single individual). Thus, the organisms within demes that construct a network with desirable characteristics receive more virtual CPU cycles than those in poorly performing demes, enabling them to replicate faster and grow to dominate the population. These studies were also conducted on larger demes, specifically 7×7 arrays.

For the majority of studies presented in this section, the AVIDA environment was configured as a grid. This means that all organisms were able to face, and thus build network links to, their 8 immediate neighbors. However, in one study (identified below), the environment topology was configured as a clique. In this case, organisms were able to face all others in their deme.

5.3.1 Necessary Homogeneity

Our first network construction experiment examined the difference between heterogeneous and homogeneous demes. Figure 5.11 plots the mean number of connected networks constructed under two different configurations across 20 trials. The first, a control experiment, allowed heterogeneous demes, while the second treatment allowed only homogeneous demes, using digital germlines. As shown, very few connected networks were constructed when demes were heteregeneous. However, when using germlines, all 20 trials were able to regularly construct connected networks. We note that when using germlines, the number of connected networks constructed at each time step continued to increase with time, indicating that the population was becoming more proficient at constructing connected networks. This result indicates that using a germline significantly increases the evolvability of coop-

erative network construction. As such, all remaining experiments in this section use digital germlines.Figure 5.12(a) is a sample network constructed by one of the dominant genotypes in this treatment. Shown here is a connected network that is missing a number of links. In this example, the missing links are likely a result of the deme being replicated as soon as the network was connected. In other examples, the missing links are the result of an evolved behavior.

5.3.2 Reducing Link Usage

In the previous experiment, while the AVIDA populations were successful at constructing connected networks, these networks contained on average 114 links – over twice as many links as what are needed to construct a connected network of 49 organisms. In this experiment we investigated the evolution of digital organisms that construct networks with reduced link usage. To encourage the construction of networks with fewer links, we augmented replicated demes with a merit that increased with fewer links. In this and subsequent experiments, AVIDA was extended with algorithms to calculate merit based on group behavior. Specifically, merit was set to the following:

$$Merit = \Big(E_{max} - |E| + 1\Big)^2 \tag{5.3}$$

where E_{max} is the maximum number of links that could be in the network (based on the size of each deme) and $|E|$ is the number of links present in the network (the additional +1 is used to differentiate between demes that replicate due to age, and those that replicate due to constructing a connected network with the maximal number of edges).

Figure 5.13 plots the mean number of links used in constructed networks, and the mean

number of connected networks actually constructed. As shown, these trials quickly evolved to use a large number of links (an effective strategy for building a connected network), and further evolution reduced the number of links that were used. At the end of this experiment, constructed networks contained an average of 86 links (1.75 links per organism), while the best performing single AVIDA trial averaged 62.1 links per network (1.2 links per organism). As before, the number of networks being constructed continues to increase with time, indicating increasing efficiency. Figure 5.12(b) depicts a sample network constructed by the dominant genotype from this experiment, while Figure 5.12(a) depicts a sample network constructed from our initial experiment where reduced link usage was not rewarded.

In examining the genomes responsible for the construction of these networks, we uncovered three primary strategies by which networks with different properties were created. First, organisms would sense their location within the environment, conditionally linking to neighboring organisms based on their location. For example, in Figure 5.12(b) we see conditional behavior based on whether the organism is along the west or north edge of the deme. Second, organisms would frequently use the rotate-$*$ instructions to unconditionally face a given direction, which when combined with location-awareness facilitates network construction. The third strategy is more stochastic in nature, where organisms would create links only if they were not already linked to at least one neighbor. Although this is an effective strategy for creating a connected network, it is difficult to construct networks with desired properties in this way. We suspect that other features of the organism's lifecycle, such as gestation time, frequency of attempting link creation, and possibly communication with neighboring organisms influenced their behavior.

5.3.3 Balancing Multiple Objectives

To this point, we have shown that digital evolution is capable of evolving cooperating organisms that construct networks while optimizing for a single criteria. Next, we investigate the evolution of organisms to construct networks that attempt to balance multiple, possibly conflicting, objectives. Specifically, here we evolve organisms that are rewarded for constructing networks that minimize both their link usage and *characteristic path length* (CPL), which is the mean pairwise distance, in number of links, between all nodes in the network. These two objectives are conflicting: A highly connected network (one with a low CPL) is likely to use a large number of links, while a network containing few links is likely to have a high CPL.

Similar to other multiobjective evolutionary optimization (MOEO) problems [144], this problem exhibits a sensitive dependence on the specific selective pressures (fitness functions, merit, etc.) that are used. We examine three different treatments are examined, each using a different formulation for the merit applied to replicated demes. Specifically, in the following *multiplicative* treatment, merit was set to:

$$Merit = \Big((CPL_{max} - CPL(N) + 1) \times (E_{max} - |E| + 1)\Big)^2 \tag{5.4}$$

The *additive* treatment is similar, changing only the internal multiplication of the CPL and link count from multiplication to addition. Finally, the *normalized* treatment extends the additive model by normalizing the CPL- and link-based terms to the range 0 to 100:

$$Merit = \left(100 \cdot \frac{CPL_{max} - CPL(N)}{CPL_{max}} + 100 \cdot \frac{E_{max} - |E|}{E_{max}} + 1\right)^2 \tag{5.5}$$

Figure 5.14 plots the mean CPL and link count for each of these three different fitness

functions. As shown, the normalized merit treatment reached its steady-state value in 10,000 updates, while the multiplicative and additive treatments continued to minimize their link usage, at the cost of an increased CPL. The different behaviors that result from these fairly minor changes to fitness functions are typical of MOEO problems, although further analysis is required to compare these results to the Pareto-optimal set. Figure 5.12(c) shows a sample network constructed by organisms attempting to balance CPL and link usage. As can be seen in this network, diagonal links were extensively used, but horizontal edges were included to further reduce CPL.

5.3.4 Reducing Diameter and Characteristic Path Length.

In some situations, such as interconnects for high-performance computing, dense communication networks are more desirable than sparse networks, as they generally lead to shorter message latencies. In this experiment, we investigate the evolution of digital organisms to construct networks that reduce two properties related to network density: characteristic path length and *diameter*, which is the maximum length of the shortest-paths between all pairs of nodes in the network, again in terms of links. To encourage reduction of diameter and CPL, we again augmented replicated demes with a merit inversely proportional to these two measures. Specifically, for the diameter-reducing treatment merit was set to the following:

$$Merit = \Big(|V| - D(N) + 1\Big)^2, \tag{5.6}$$

where $|V|$ is the number of organisms present in the network and $D(N)$ is the calculated diameter of the network. Similarly, for the CPL-reducing treatment, merit was set to the

following:

$$Merit = \Big(CPL_{max} - CPL(N) + 1\Big)^2, \tag{5.7}$$

where $CPL(N)$ is the calculated CPL of the network and CPL_{max} is the maximum possible value of the CPL (given our knowledge of the underlying environmental geometry and size of the demes).

Figure 5.15 plots the mean diameter and mean CPL for these two experiments. As can be seen here, both diameter and CPL approach their optimal values (for the spatial grid used in these experiments, the optimal diameter is 6, while the optimal CPL is 3.5). In each experiment, the minimization of diameter and CPL come at the expense of link usage, which averaged 138. In an experiment described later, we attempt to balance such competing concerns.

5.3.5 Clustering Coefficient

The *clustering coefficient* of a network is a measure of connectedness. Formally, the clustering coefficient is the ratio of the number of links between nodes in a given node's neighborhood to the number of possible links between those nodes, averaged over all nodes in the network. In a sense, it measures how many neighbors of a given node are themselves neighbors of each other. The clustering coefficient has been used to characterize *small-world* and *scale-free* networks, which have been found to describe network structure such as the branching of trees and the Internet [145]. In this experiment, we investigate the evolution of digital organisms to construct networks that exhibit specified clustering coefficients.

One difference between this experiment and all others is that the underlying environmental geometry was changed from a spatial grid to a clique. This change enables any organism

to link to any other organism within the same deme, and is necessary to reach higher clustering coefficients. To encourage the maximization of the clustering coefficient, merit was set to the following function:

$$Merit = \Big(100.0 \cdot CC(N)\Big)^2 \tag{5.8}$$

where $CC(N)$ is the calculated clustering coefficient of the constructed network. To minimize the clustering coefficient of connected networks, merit was set to:

$$Merit = \Big(101.0 - 100.0 \cdot CC(N)\Big)^2 \tag{5.9}$$

We also wished to evolve organisms that constructed networks with an arbitrarily-selected clustering coefficient of 0.5. For this purpose, we set merit to:

$$Merit = 0.01^{2 \cdot |0.5 - CC(N)|} \tag{5.10}$$

Figure 5.16 plots the three different mean clustering coefficients for networks produced by populations evolved with the deme merit functions described above. Here we see that each treatment reached its steady-state value within 20,000 updates, with the minimizing treatment reducing the clustering coefficient to 0.2, the maximizing treatment increasing the clustering coefficient to 0.99, and the targeted treatment produced a clustering coefficient of 0.57, an error of approximately 7% from the requested clustering coefficient. Figure 5.17(a) and Figure 5.17(b) depict sample networks constructed by the dominant genotypes from the minimizing and targeted experiments. Considering the importance of communication

networks to distributed systems, it is a promising result that AVIDA is able to evolve populations of digital organisms that are capable of constructing networks that meet complex criteria.

Each of the studies presented thus far has focused on a characteristic of structural behavior in isolation. We have seen how the underlying network may affect the evolution of behavior in digital organisms, and we have also seen how digital organisms can evolve to construct networks with different properties. In the next section, we investigate the joint evolution of network construction and data distribution.

5.4 Construction of Networks for Data Distribution

In this section we present the results of our experiments using several different fitness functions and experimental setups for jointly evolving network construction and data distribution. We first examine a scenario where the fitness function includes both network and data distribution components. Next, we remove the network construction component from the fitness function, allowing evolution to discover the network construction implicitly. In our final experiment, we turn our attention to various network faults and explore the effects of link decay and node turnover on the evolution of these two behaviors. We conclude this section with a discussion of these experimental results, specifically focusing on two questions: First, is there evidence to support the conclusion that network construction behaviors in particular are adaptive? In other words, is selection *refining* network construction behaviors, or are we discovering degenerate networks, e.g., completely connected networks, that are merely sufficient for the data distribution task to be solved? Second, we examine a number of the strategies that evolved in the presence of link decay and node failure.

Table 5.3: 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 a 5×5 grid of cells
Environment topology	Grid; organisms can face (and build links to) all others in their deme
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, probabilistic scheduler
CompeteDemes	Compete all demes using (1,2) tournament selection (periodic, every 400 updates).

Methods. For these experiments we return to using the CompeteDemes process and germlines, with specific configuration values summarized in Table 5.3. We used 400 demes, with each deme configured to use a germline to provide homogeneity within demes. We configured the environment such that organisms could face (and thus build logical links to) all organisms in their deme, while maintaining an underlying spatial grid structure. We also return to demes of 25 cells, the default values were used for all mutation rates, and the *time slice* was set to 5 instructions per update.

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 CompeteDemes process, the offspring deme was filled with 25 copies of the latest (possibly mutated) genome from the germline. As in our previous studies of network construction, deme replication does *not* include the logical network; offspring demes are initialized with an empty network. As the final step of Avida configuration, we defined the default ancestor to contain 100 nop-C instructions.

5.4.1 Multi-objective network construction

Our first experiment focused on whether we could discover digital organisms that were able to optimize the structure of their communication network while trying to solve the data distribution task described in Section 5.1. Specifically, we defined a fitness function that rewarded first for solving the data distribution task, and then further rewarded for the construction of least-cost networks, where the cost of a network is defined as the sum of Euclidean link lengths between cells in spatial environment. Specifically, we defined the fitness function F for each deme as:

$$F = \begin{cases} (1+R)^2 & \text{if } R < 25 \\ (1+R+1000.0/S)^2 & \text{if } R = 25 \end{cases} \tag{5.11}$$

where R is the number of organisms within the deme that indicate they have received data (via the set-opinion instruction) and S is the sum of the Euclidean link lengths. For example, in Figure 5.2(b), $S = 16.46$ (8 links of length 1, 6 diagonal links of length 1.41). Here, we use Euclidean link lengths as a generic cost metric; it could easily be replaced with a measure of network congestion, delay, or bandwidth.

We conducted four different treatments of this fitness function: a *broadcast* treatment, a *unicast* treatment, and a control for each. In the broadcast treatment, only the network-bcast1 instruction was available for sending messages, while in the unicast treatment, only the network-unicast instruction was available. The two control treatments were configured identically, except that the winner of each tourney was selected randomly instead of by fitness.

Figure 5.18(a) plots the grand mean deme fitness over time across all 30 trials for these

four treatments (in this plot and all following, error bars are 95% confidence intervals conducted via 200-fold bootstrapping). As shown in this figure, the broadcast treatment results in a statistically significant higher mean fitness than the unicast treatment (Wilcoxon rank sum test, $p < 0.001$), although individual trials from both treatments achieved the maximum possible fitness at different points. Figure 5.18(b) is a box plot of the final values from these two treatments, which more clearly shows the final fitness differences between these treatments. This result shows that the available network primitives can affect network structure. In this case, we see that the availability of message broadcasting enabled the construction of lower-cost networks. We found no evidence of control treatments solving the data distribution task, indicating that this behavior is not the result of genetic drift.

5.4.2 Implicit selection of network construction

The goal of our next experiment was to determine if evolution was able to optimize a "hidden" goal. In other words, could evolution optimize the underlying network while only rewarding for a communication task? Specifically, we defined a fitness function that rewarded first for solving the data distribution task, and then further rewarded for a reduction in the number of messages sent during the competition period. We defined the fitness function F for each deme as:

$$F = \begin{cases} (1+R)^2 & \text{if } R < 25 \\ (1+R+25,000/M)^2 & \text{if } R = 25 \end{cases} \tag{5.12}$$

where R is the number of organisms within the deme that indicate they have received data, and M is the total number of messages received by all organisms within the deme (for parity among treatments, a broadcast is counted once for each *receiving* organism). We conducted

the same four different treatments of this fitness function as in the previous experiment (broadcast, unicast, and their respective controls).

Figure 5.19(a) plots the grand mean deme fitness across all 30 trials for these four treatments. As shown here, the broadcast treatment again results in a statistically significant higher fitness than unicast ($p = 0.0184$), and the highest-performing deme sent only 28 messages (the highest-performing deme from the unicast treatment sent 31). Figure 5.19(b) is a box plot of the final values from each treatment. In this case, we see that including a broadcast capability enabled the evolution of behaviors that used fewer messages than in unicast-only treatments.

5.4.3 Network decay

In our final series of experiments, we examined the effects of various network hazards, specifically link and node decay. We first tested various rates of link decay, where links would be removed from the communication network based on their age. Next, we added varying rates of node decay, where nodes (and all connected links) would be removed with a configurable probability, *and* links decayed based on their age. When a node is removed, we replace the corresponding digital organism with a new one constructed from the deme's germline. In all treatments, we used the fitness function from Equation 5.12, the same as in the previous experiment.

Figure 5.20 depicts the results of two different treatments. In Figure 5.20(a), we see the grand mean deme fitness for three different link decay rates (5, 10, and 20 updates), plotted as lines (d5, d10, and d20, respectively); a control that does not include decay is also included. Here we see that the presence of link decay unilaterally improves fitness, a

somewhat non-intuitive result given the corresponding increase in difficulty of solving the data distribution task. In Figure 5.20(b), we see the grand mean fitness for these same link decay rates *combined with* a node decay rate of 10% (the control treatment is node decay without link decay). At this rate of node decay, approximately 40 nodes (and their corresponding digital organisms) decay during each deme competition period. In this figure, we see that node decay dramatically reduces fitness (note scale change), and, interestingly, that the presence of link decay improves fitness in this case. Additional rates of node decay were tested (1%, 5%, and 2%); results were qualitatively similar, though less pronounced. These results indicate that the presence of link decay appears to have made possible the evolution of highly-efficient strategies for data distribution, which we examine in more detail in the next section.

5.4.4 Evolved behaviors

In the previous sections, we presented results of experiments using digital evolution to discover network construction and data distribution behaviors in the presence of link and node decay. Here we describe some of the common strategies that were evolved. For brevity, we have named these strategies *typewriter*, *echo*, *active-listener*, and *active-echo*; these strategies are described below. The first three strategies were common approaches to network construction and data distribution in the presence of link decay only, while the active-echo strategy was commonly found in the presence of both link and node decay. In general, different rates of link and node decay had only marginal effects on the evolved behavior, usually in the form of changing rates of periodic broadcasts; the accompanying illustrations to the strategies described below are from treatments with a link decay of 10 updates; the

active-echo strategy also included 10% node decay.

Typewriter. As with many of the evolved strategies, the typewriter, named for its sequential typewriter-like strategy, involves multiple distinct behaviors: First, all organisms periodically attempt to retrieve messages from their message buffer, and if a message was received, set their opinion to the data contained therein. Second, the owner of the cell data (the organism inhabiting that cell) iterates through all organisms in the deme, connecting to them and sending a message in turn. This process is illustrated in Figure 5.21. In this and following figures, a grid is used to represent the cells in a deme, and cells are always occupied by digital organisms. Circles and lines are used to represent nodes and links in the constructed network, respectively. The color of the node indicates whether the node is the source of a link (red), destination of a link (blue), or is sending a message (green). Similarly, the color of a link represents whether the link is being used to send a message at that point in time (green), or not (black). Figure 5.21(a) shows the initial state of the network, with no nodes or links yet constructed. In Figures 5.21(b)-(d), we see the cell data owner (green) iterate through the cells in the deme. Specifically, for each other cell in the deme, the cell data owner first establishes a connection to the occupant, sends the cell data, and then allows that link to expire as it proceeds to the next cell.

Echo. Figure 5.22 shows the echo strategy, which has three behavioral components: First, all organisms create a link to the same "root" cell, building a depth-1 tree (the specific cell that is used for the root is likely a consequence of genetic drift, although cell 0 is common). The second behavior is that any messages are echoed shortly after being received. The final behavior, performed only by the cell data owner, is to send the cell data after it has built

the link to the root. The combination of these three behaviors is a reliable solution to the network construction and data distribution problem regardless of which cell actually contains the cell data. This strategy is depicted in Figure 5.22. In Figure 5.22(a), we see organisms throughout the deme all connecting to the root in the upper-left. Next, in Figure 5.22(b), not only have all organisms connected to the root, we also see a single organism transmitting a message containing the cell data to the root node. Then, in Figure 5.22(c), only 2 updates later, the root node echos this message to all those that had previously connected to it. Finally, Figure 5.22(d) shows the echo from all organisms to the broadcast message that was sent by the root node. This strategy is an excellent example of the role that network construction can play in distributed applications, where the relationship of message sending behavior to the topology of the constructed network enables efficient messaging. In some variants of this strategy, links are continually refreshed (though unused), while in others the links are left to decay.

Active-Listener. Figure 5.23 shows the active-listener strategy, which has two distinct behaviors. First, all organisms iteratively connect to all other cells in the deme, approximately in the same order. Second, organisms that either occupy the cell containing the cell data or have received it via a message periodically broadcast the cell data. Finally, organisms that are connected during such a broadcast change their behavior from iteration to broadcast. In this way, only those individuals that have not yet received information initiate network connections, while those that have the cell data simply broadcast it for others. Figure 5.23(a) and (b) show part of this iterative process, where the majority of organisms connect to the cell in the lower-middle, and then connect to the cell in the lower-left. Figure 5.23(c) and (d) then show organisms with the cell data broadcast to those that have

initiated network connections.

Active-Echo. Figure 5.24 shows the active-echo strategy, which was evolved in the presence of both link and node decay. This strategy contains components of both the echo and active-listener strategies previously described. In Figure 5.24(a)-(c) we see similar behavior to the active-echo strategy, where all organisms connect to a root node that broadcasts the cell data. Figure 5.24(d) depicts the active-listener component of this strategy, where organisms periodically broadcast the cell data while the network is allowed to decay. We note that whenever a node fails, its subsequent replacement follows the active-listener strategy, linking to random nodes within the deme until it receives a message from one of the periodic broadcasts. The newborn then changes its behavior and begins periodic broadcasting of its own. This strategy thus relies on a combination of rapid data distribution early in the deme competition period, followed by a stochastic period during which new organisms are incorporated.

5.4.5 Adaptivity of Network Construction

We have presented results of experiments in the joint evolution of network construction and data distribution behaviors. Based on two different fitness functions, we have evolved behaviors that both build networks, and then use those networks for data distribution. However, fitness is a relatively simple measure, and it tells us little about the characteristics of the constructed networks. For example, it could be the case that the constructed networks are degenerate in some way, e.g., are they completely connected, or are they similar to the networks that would be built via genetic drift? In other words, we wish to discover if the network construction behavior is adaptive, in the sense that it can provide a selective

advantage.

Figure 5.25 contains box plots of different network metrics, drawn from the final generation of both broadcast and unicast treatments and their controls. Figure 5.25(a) plots the fraction of connected (a path exists between all pairs of nodes) networks for each of these treatments. As seen here, the broadcast and unicast treatments have a statistically significant higher proportion of connected networks than the controls (Wilcoxon rank-sum test, $p < 0.0001$ between treatments and controls; $p < 0.005$ between treatments). This result would appear to indicate that network construction behaviors are adaptive. Another possibility, however, is that the network construction behavior is degenerate. For example, the digital organisms could simply be building a completely connected network, where all pairs of nodes are linked. Figure 5.25(b) plots the number of edges used in constructed networks. Again, we see that the broadcast and unicast treatments are statistically significantly different from the control treatments ($p < 0.0001$ in all cases). Furthermore, the number of edges being used in both broadcast and unicast are significantly less than would appear in a completely connected network (25 nodes, 300 edges). Figure 5.25(c) plots the characteristic path length (CPL); in general, tree-structured networks will have a CPL equal to just under twice the tree depth, while completely connected networks will have CPL of 1.0. As shown in this figure, the average CPL of broadcast and unicast networks are near 2.0, indicating a tree of depth 1. Broadcast and unicast treatments were again statistically significantly different from controls ($p < 0.0001$) and each other ($p < 0.05$). Finally, Figure 5.25(d) plots the clustering coefficient. Intuitively, clustering coefficients near 1.0 indicate a highly connected graph with many large clusters of nodes, while clustering coefficients nearer to 0.0 indicate a less well-connected graph with few clusters. Broadcast and unicast treatments were again

statistically significantly different from controls ($p < 0.0001$). Based on this evidence, we conclude that network construction is an adaptive behavior for the multi-objective fitness function.

Figure 5.26 depicts the same network metrics for the implicit selection case, again over broadcast, unicast, and control treatments. Figure 5.26(a) plots the fraction of connected networks among all treatments, and as seen here, the broadcast and unicast treatments again have a significantly higher fraction of connected networks than the controls ($p < 0.0001$). Figures 5.26(b)-5.26(d) depict the number of edges,CPL, and clustering coefficient, respectively. In all cases, the broadcast and unicast treatments were statistically significantly different from the control treatments ($p < 0.01$) and each other ($p < 0.0001$). Interestingly, the networks constructed by the unicast treatment under this fitness function contain a significantly greater number of edges than under the previous treatment. In fact, the unicast networks are nearly completely connected. Furthermore, clustering coefficient in the broadcast treatment varies significantly more here than in the multi-objective case. This indicates that a wide variety of different network topologies are being explored, in contrast to the multi-objective case where the clustering coefficient varied little.

Based on the results of these analyses, it appears that the network construction behaviors are subject to selection, and are adapting to the fitness function being used. Specifically, network metrics such as characteristic path length and clustering coefficient vary not only among treatments, but also across fitness functions, and they are statistically significantly different from control treatments where network construction behaviors were not under selection. These studies demonstrate that digital evolution, and evolutionary algorithms more generally, can discover network construction behaviors implicitly, based purely

on application-specific fitness functions, which suggests that EAs could be an effective tool for discovering network construction algorithms tuned for specific distributed applications.

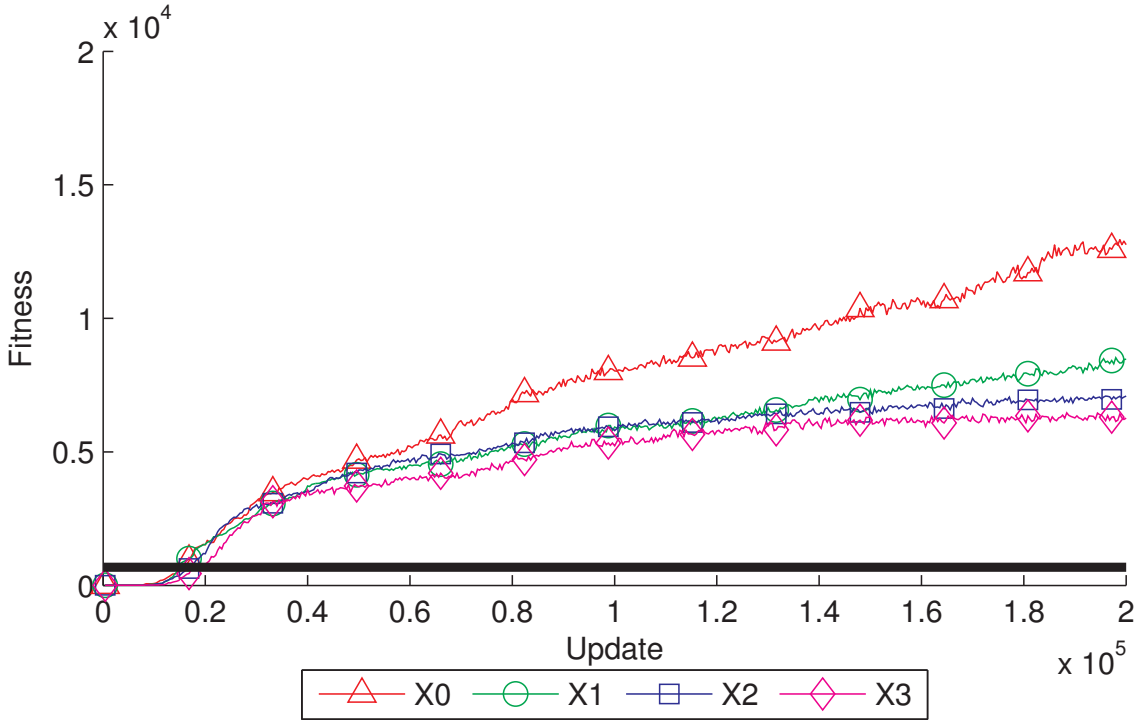
5.5 Conclusion

In this chapter, we have presented the results of a series of studies of structural behavior, specifically the use and construction of communication networks. In our first study, we explored the effect that different network topologies may have on the evolution of data distribution, a behavior in which data was to be distributed within a group. We found that groups of digital organisms were able to solve this problem, even in the presence of failures.

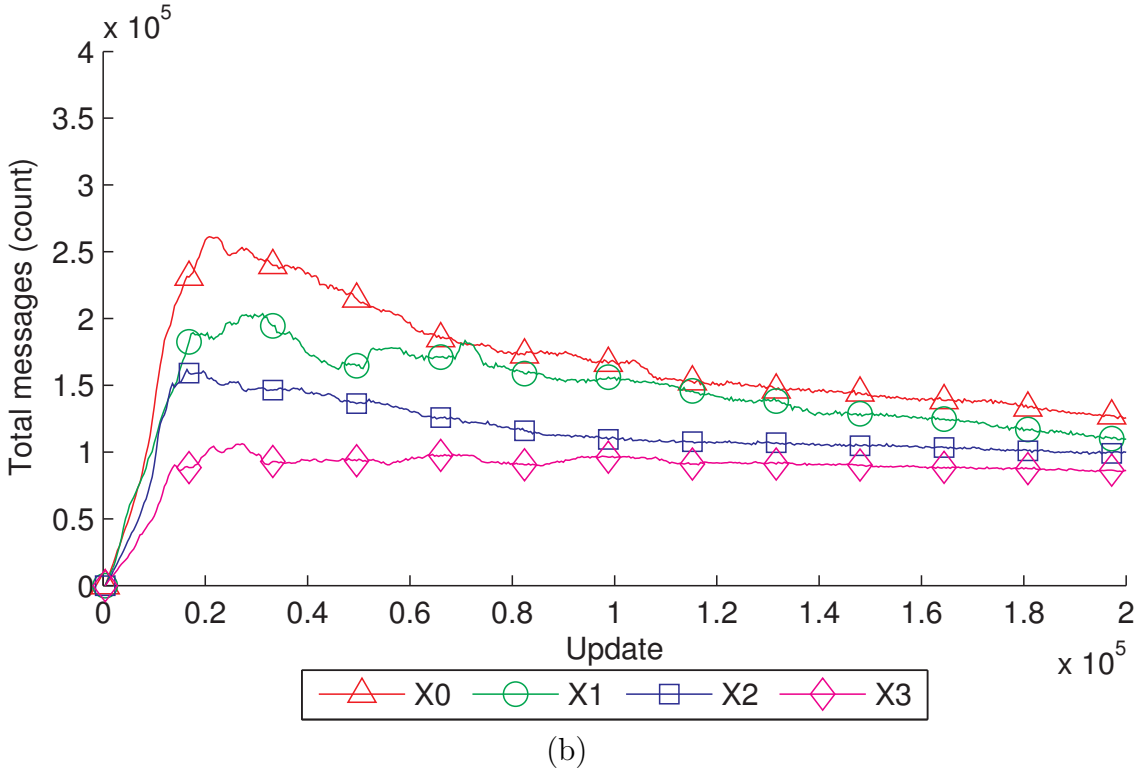
Next, we enabled organisms to construct networks in Avida. By applying selective pressures that were based on the structure of these networks, we were able to evolve groups of organisms that constructed networks that varied in complexity from simple networks that used few links, to more complex networks that had an arbitrary clustering coefficient.

In our final study, we investigated the evolution of joint behaviors for network construction and data distribution, and we found that Avida was able to evolve digital organisms that efficiently solved the data distributed problem while constructing networks tuned specifically for that task. This result hints at a new class of distributed algorithm that automatically builds a network specifically tuned for its own needs. This capability could be especially useful in hazardous and/or dynamic environments, such as mobile sensor networks.

To this point, we have seen how Avida is able to evolve various classes of distributed behavior in isolation. We now turn to a study of the evolution of a compound behavior, where we simultaneously evolve data-driven, temporal, and structural behaviors. Specifically, we investigate the evolution of control algorithms for a simulated mobile ad hoc network.



(a)



(b)

Figure 5.10: Fitness (a) and message counts (b) for a range of values for message loss, message cost, and node churn.

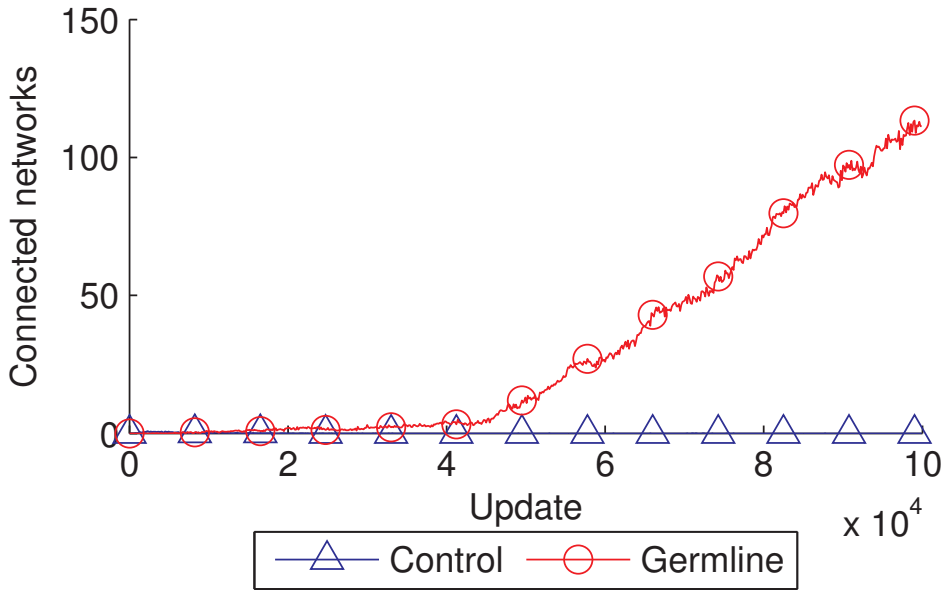
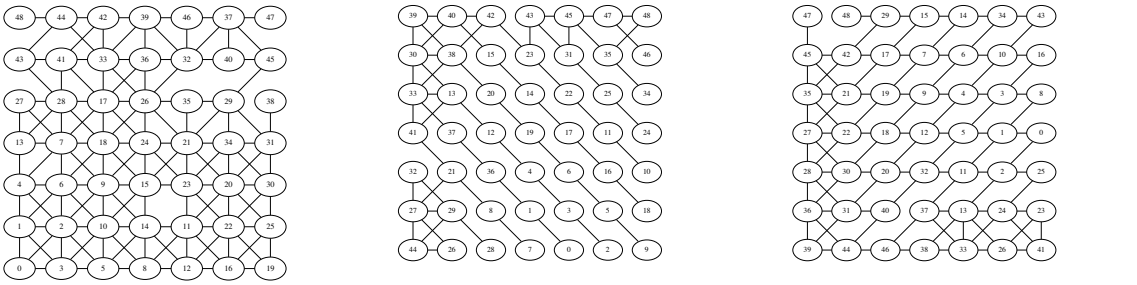


Figure 5.11: Mean number of connected networks constructed under heterogeneous (control) and homogeneous (germline) configurations, per 100 updates. Digital germlines were significantly more effective at evolving network construction.



(a) connected network (b) reduced link usage (c) balancing CPL and link usage

Figure 5.12: Example networks constructed by evolved organisms.

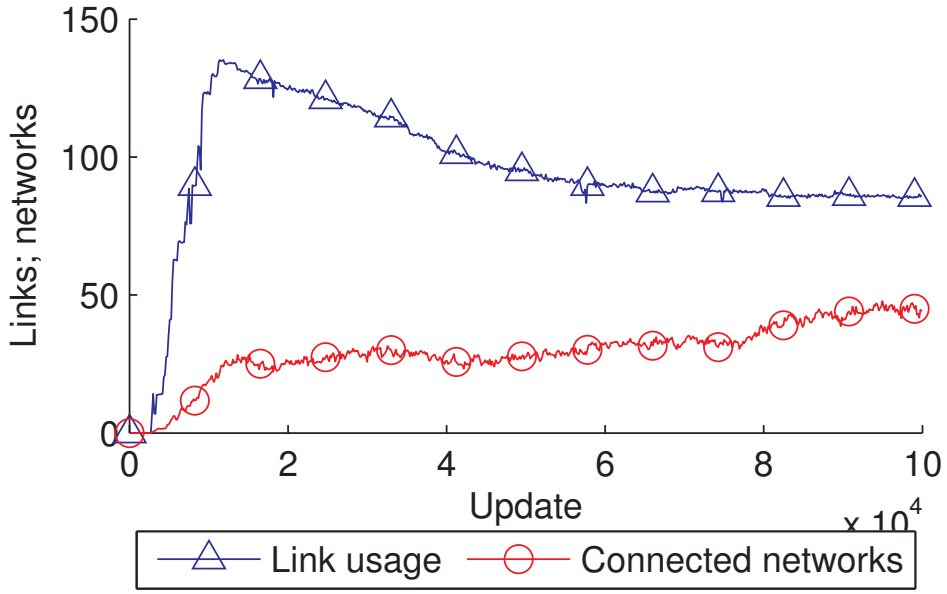
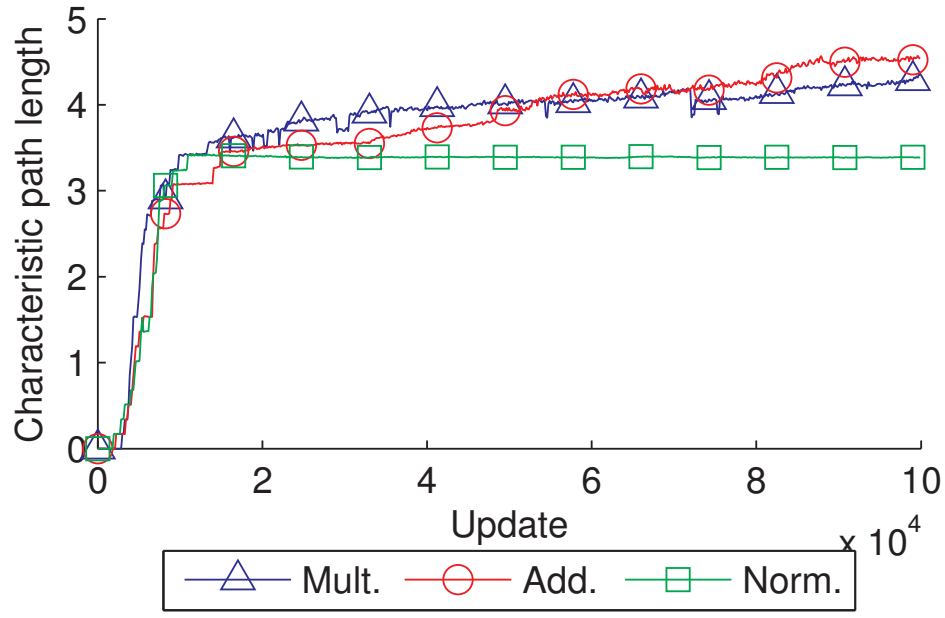
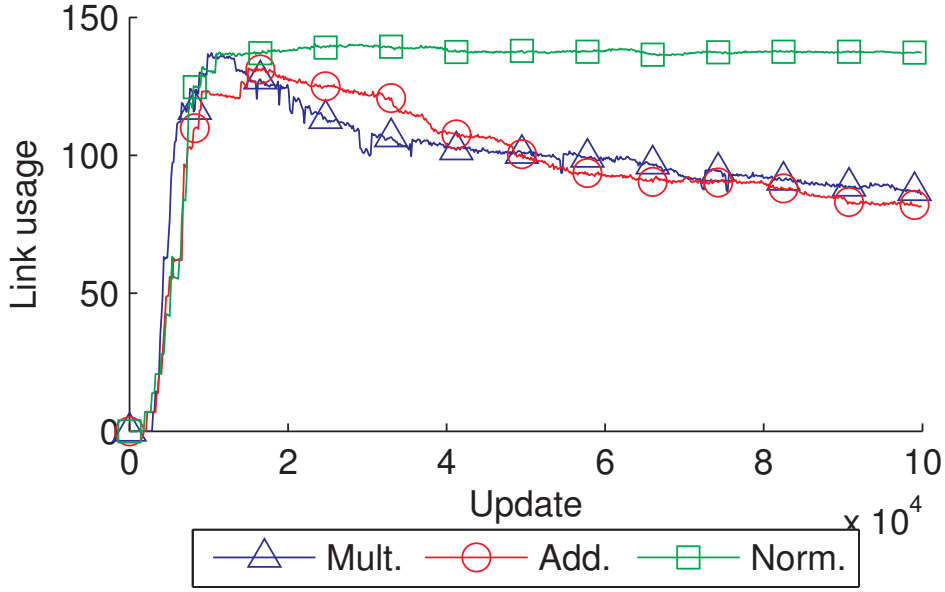


Figure 5.13: Mean number of links used to construct networks, per 100 updates. Selecting for networks with fewer links results in organisms that use links sparingly, while still building a connected network.



(a)



(b)

Figure 5.14: Mean characteristic path length (CPL) (a) and link usage (b) under three different experiments that balance CPL and link usage, per 100 updates.

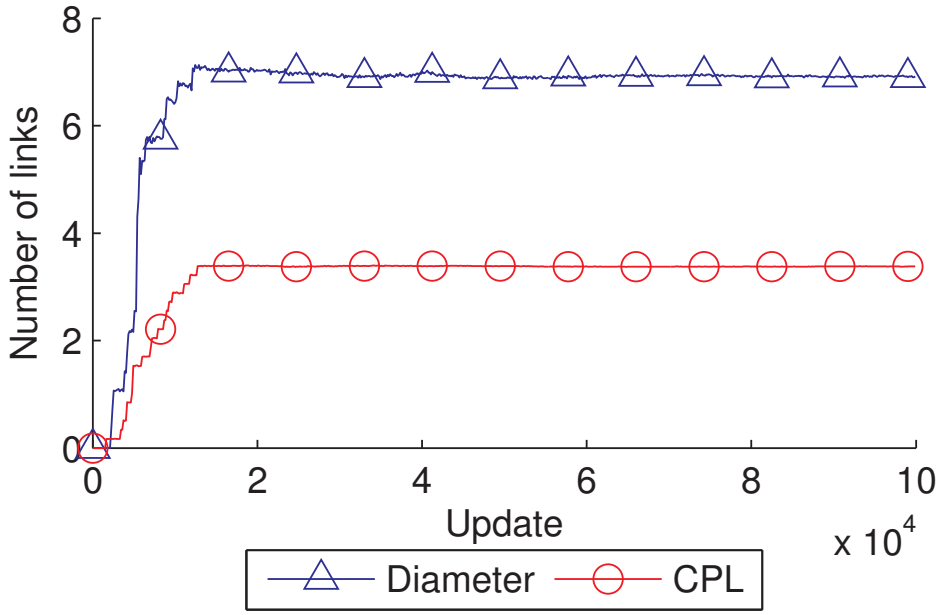


Figure 5.15: Mean number of links for diameter-reducing and characteristic path length (CPL)-reducing experiments, per 100 updates. In both cases, organisms evolved to construct networks that approached their optimal diameter and CPL.

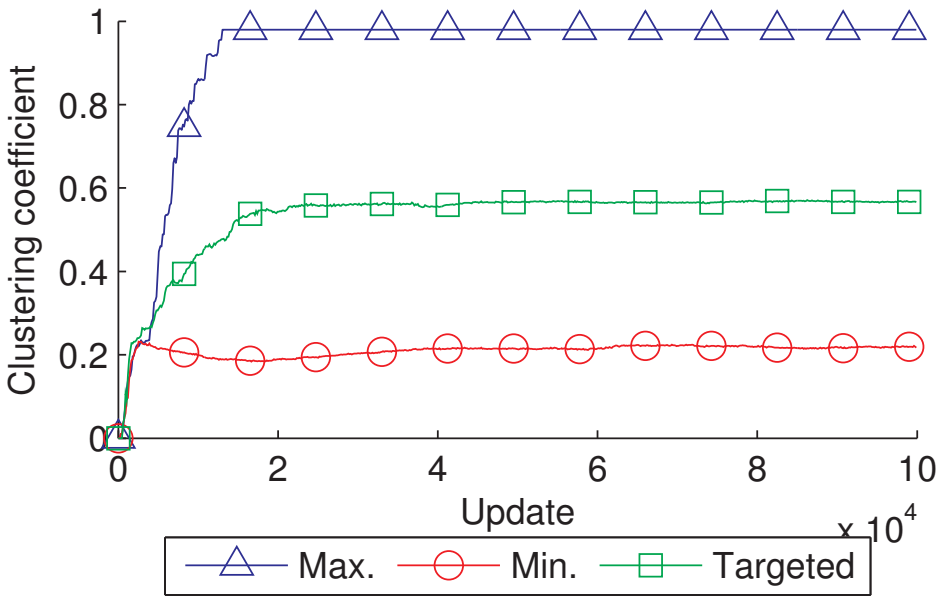
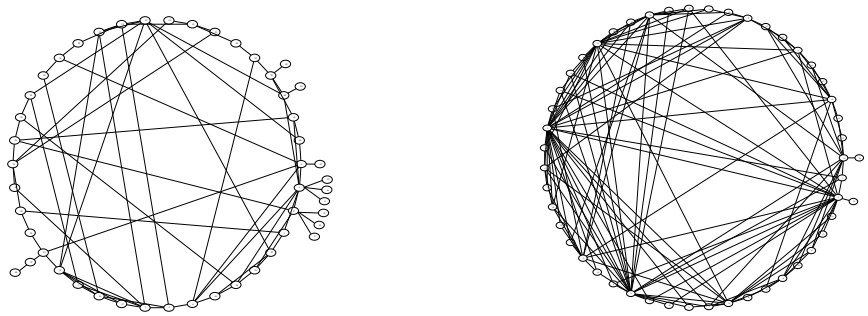


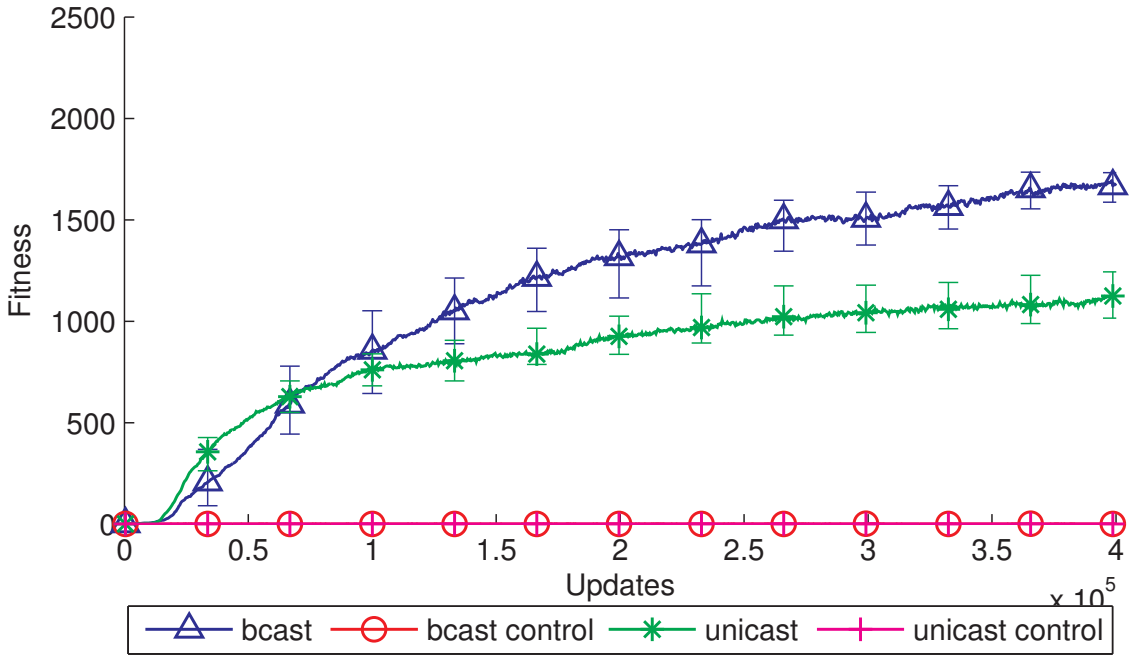
Figure 5.16: Mean clustering coefficients for three different experiments, per 100 updates. Organisms evolved to construct connected networks with maximal, minimal, and targeted (0.5 desired, 7% error) clustering coefficients.



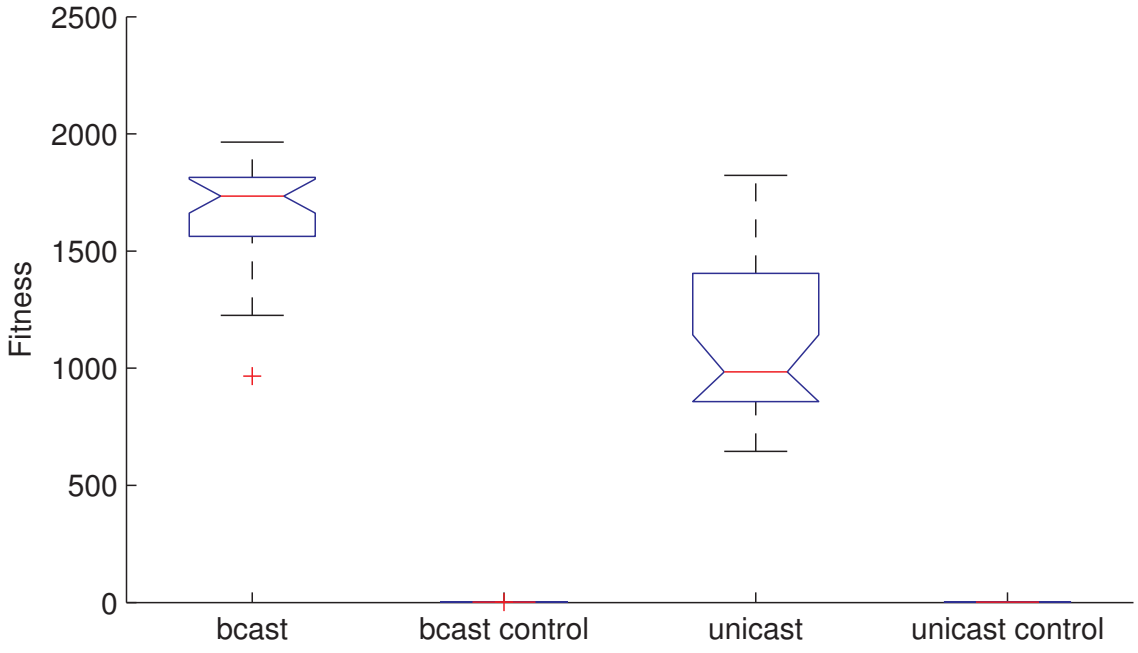
(a) minimizing clustering coefficient

(b) targeted clustering coefficient

Figure 5.17: Example networks constructed by evolved organisms.

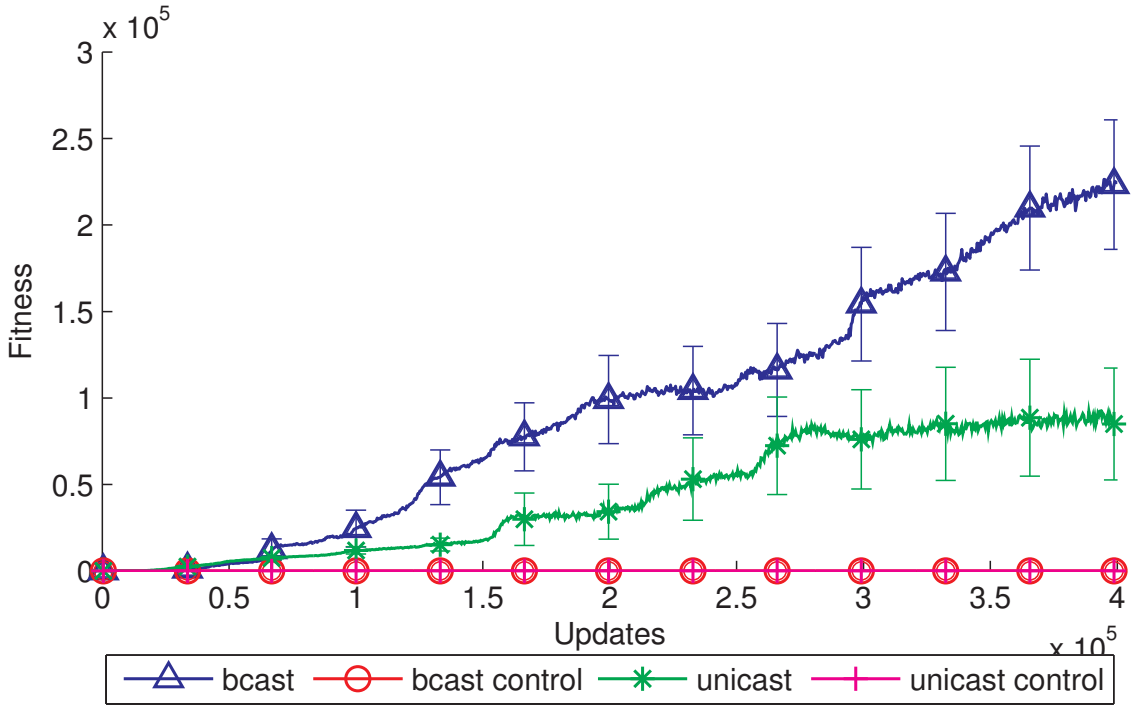


(a)

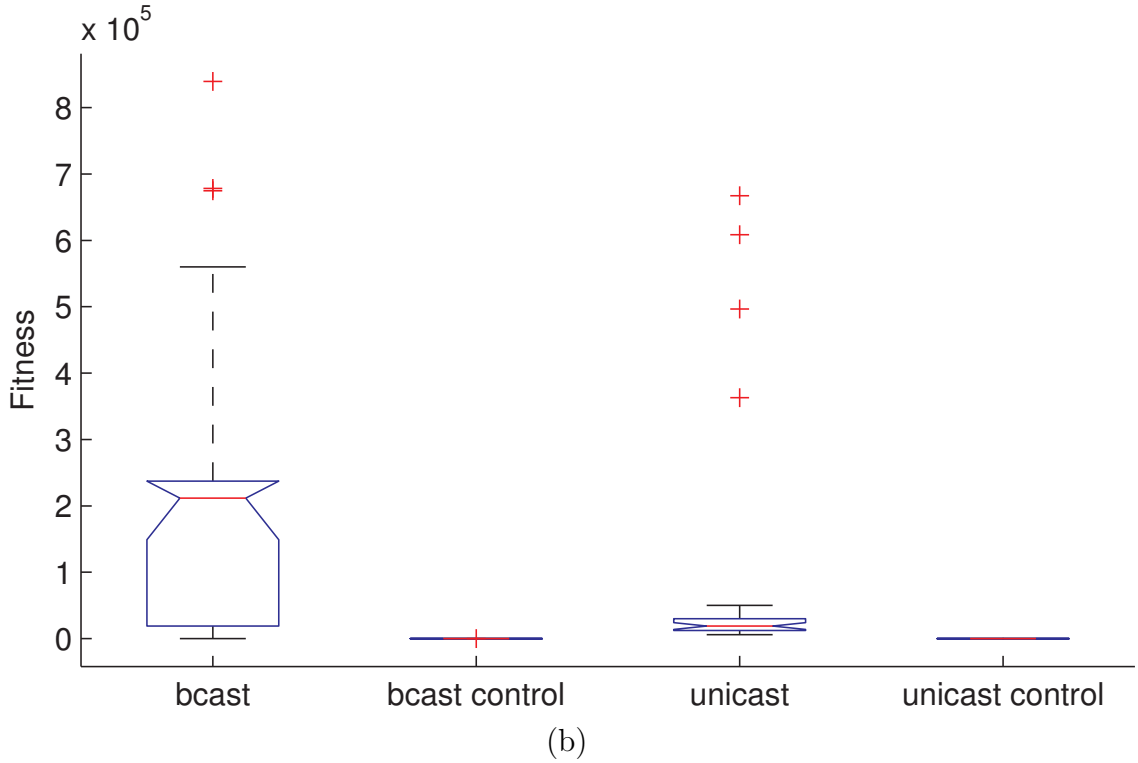


(b)

Figure 5.18: Grand mean deme fitness over time for the broadcast, unicast, and control treatments (a), and a box plot of final fitness values (b). The availability of a broadcast instruction enabled the construction of lower-cost networks.

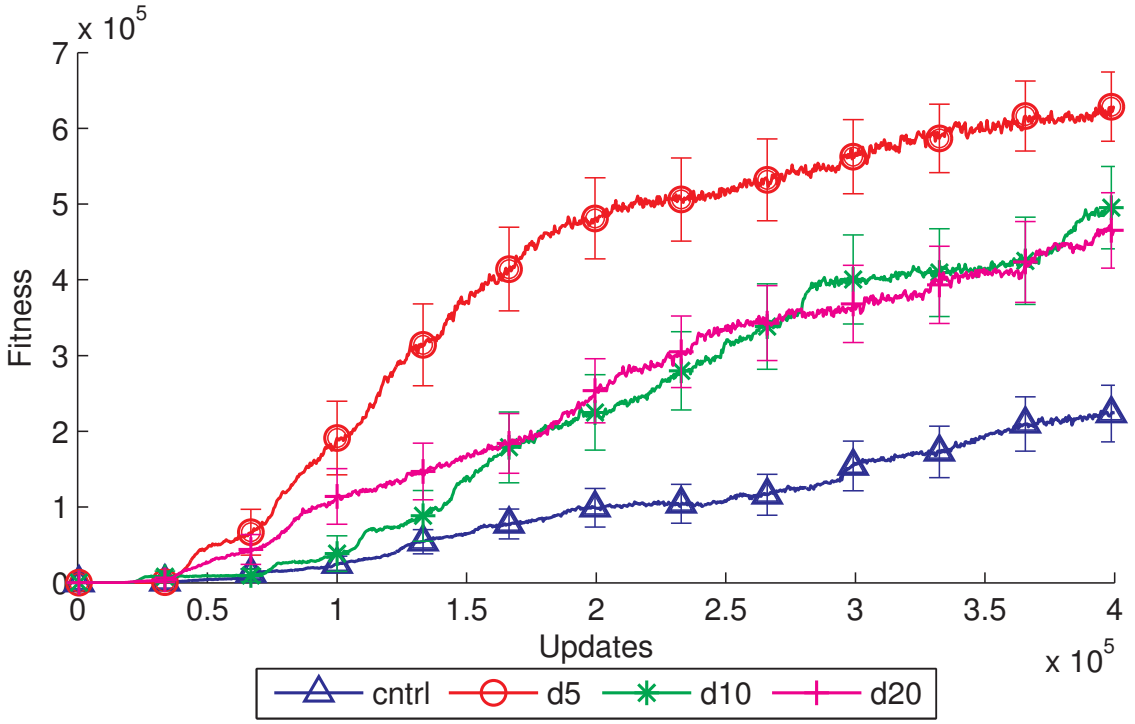


(a)

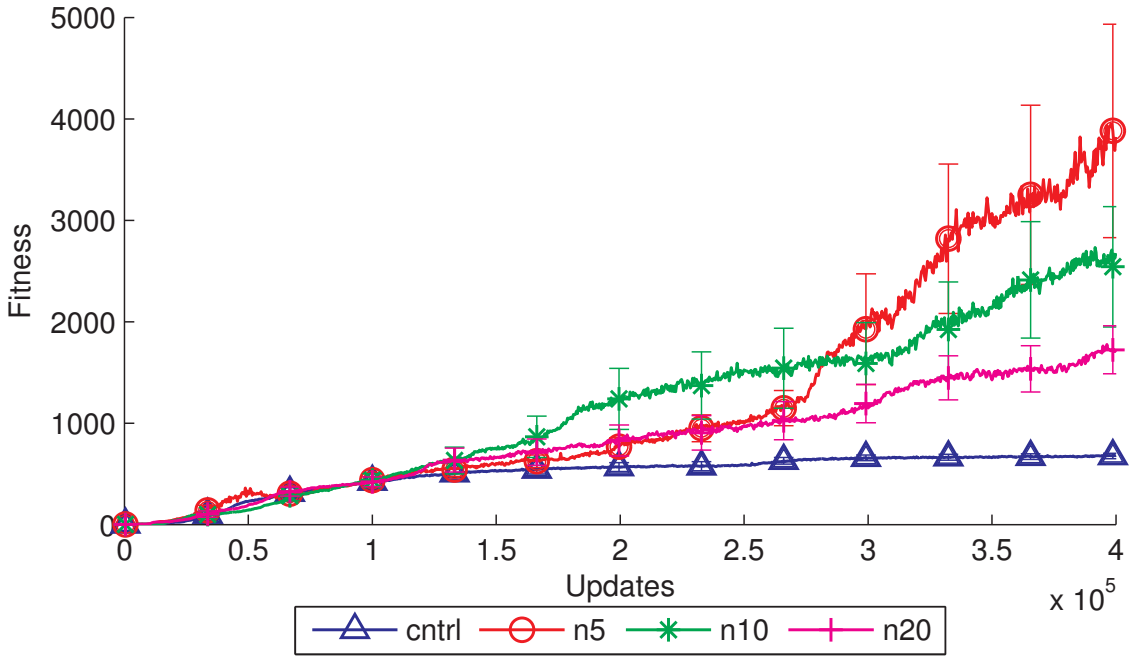


(b)

Figure 5.19: Grand mean deme fitness over time for the broadcast, unicast, and control treatments (a), and a box plot of final fitness values (b). The availability of a broadcast instruction enabled the evolution of more efficient communication behavior.

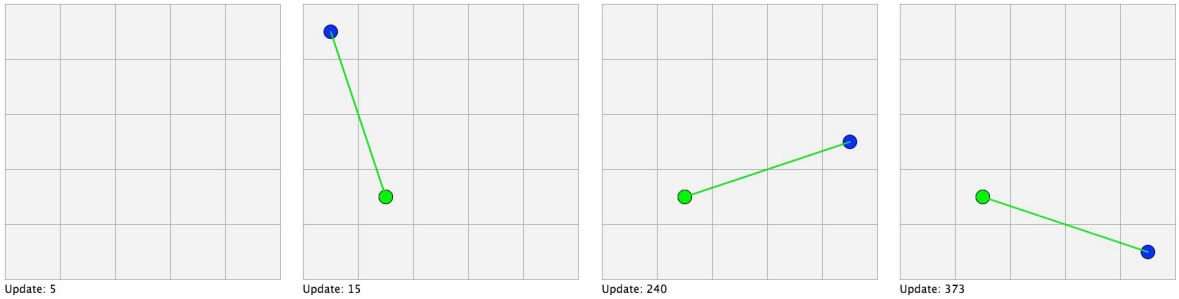


(a) Three different link decay rates (5, 10, and 20 updates) plotted as lines (d5, d10, and d20), respectively; cntrl is a control that does not include link decay.



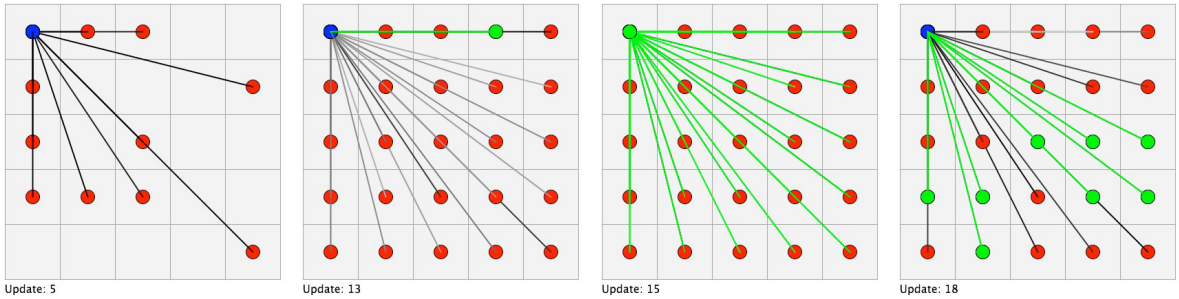
(b) Node decay of 10% combined with three different link decay rates (5, 10, and 20 updates) plotted as lines (n5, n10, and n20), respectively; cntrl includes only node decay.

Figure 5.20: Grand mean deme fitness in the presence of varying rates of link decay (a), and varying rates of link decay combined with a 10% node decay rate (b).



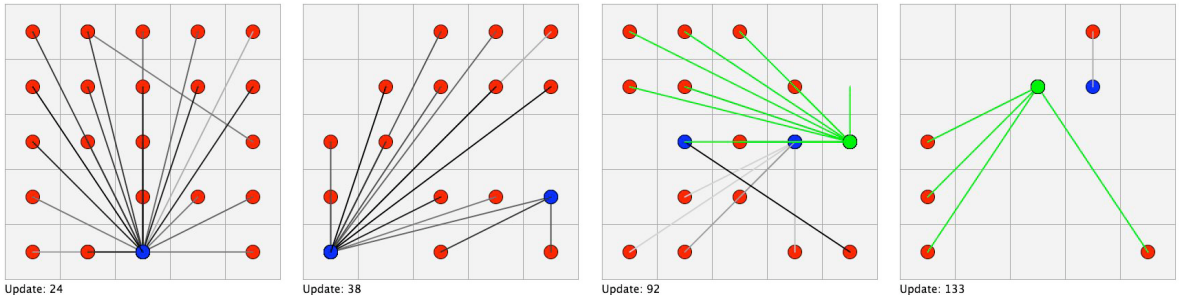
(a) (b) (c) (d)

Figure 5.21: Example of the *typewriter* strategy. Beginning with an empty network (a), the organism occupying the cell containing the cell data (green circle) iteratively connects and sends a message to other organisms in the deme (b)-(d).



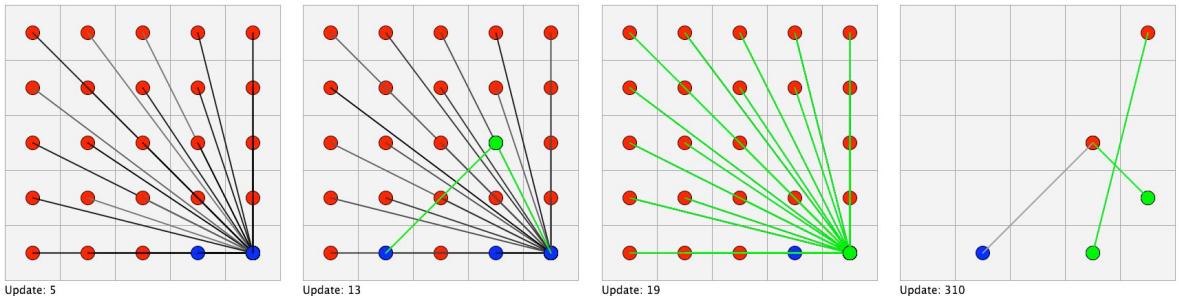
(a) (b) (c) (d)

Figure 5.22: Example of the *echo* strategy. All organisms connect to the root node (blue circle, upper-left), which, after receiving the cell data in (b), broadcasts it to all connected organisms (c), who then echo the message in (d).



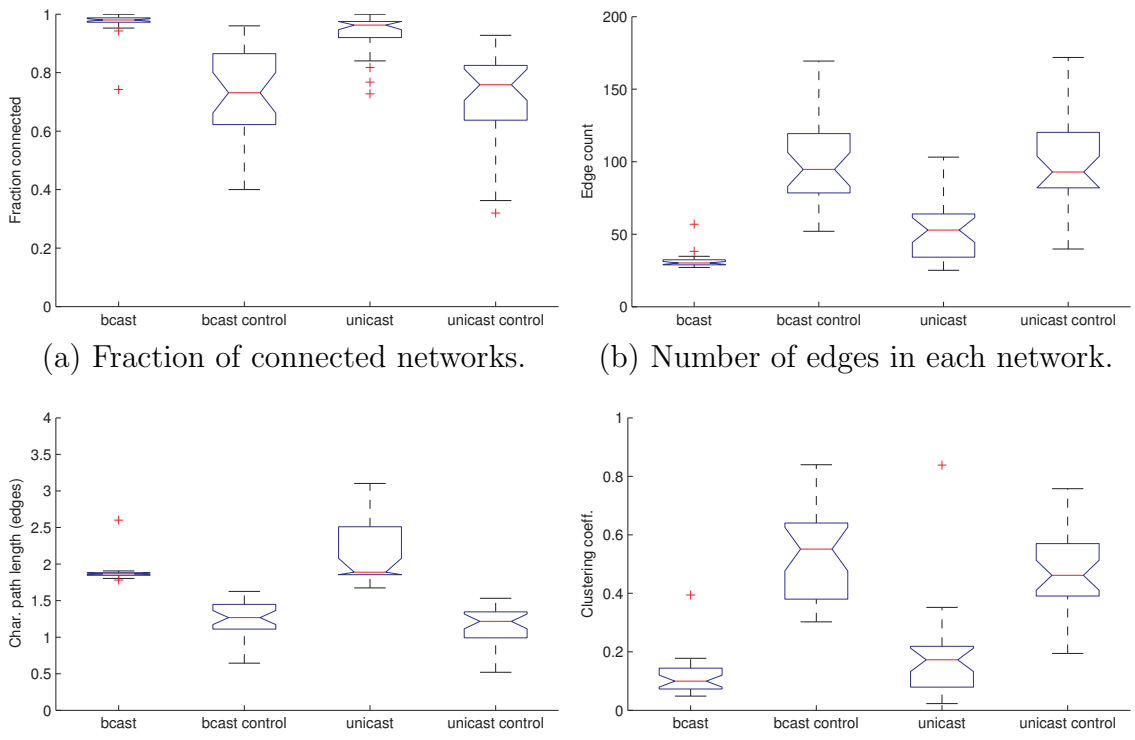
(a) (b) (c) (d)

Figure 5.23: Example of the *active-listener* strategy. All organisms iteratively connect to cells in the deme (a)-(b), and then organisms with the cell data broadcast to those that have initiated network connections (c)-(d).



(a) (b) (c) (d)

Figure 5.24: Example of the *active-echo* strategy. All organisms connect to a root node which broadcasts the cell data (a)-(c), after which point new organisms receive the cell data via a stochastic process (d).



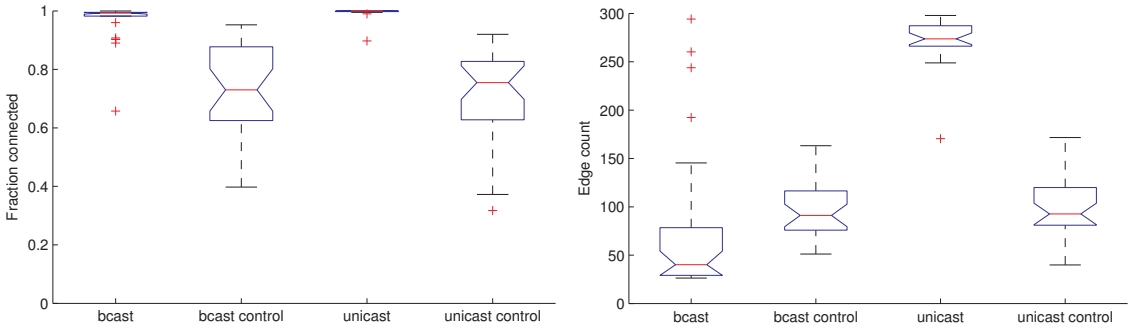
(a) Fraction of connected networks.

(b) Number of edges in each network.

(c) Characteristic path length.

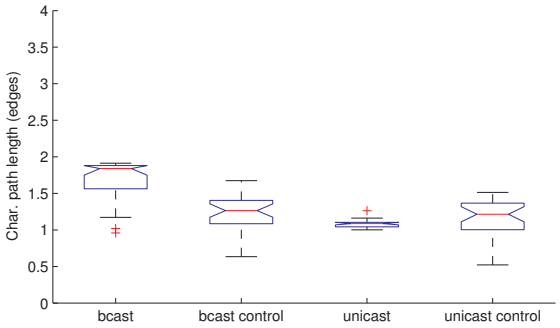
(d) Clustering coefficient.

Figure 5.25: Different network characteristics calculated on the networks constructed under the multi-objective fitness function from Section 5.4.1. In all cases, broadcast and unicast treatments differ from controls at the $p < 0.01$ level.

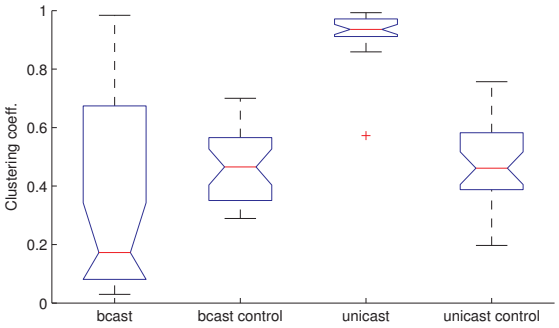


(a) Fraction of connected networks.

(b) Number of edges in each network.



(c) Characteristic path length.



(d) Clustering coefficient.

Figure 5.26: Different network characteristics calculated on the networks constructed under the implicit selection fitness function from Section 5.4.2. In all cases, broadcast and unicast treatments differ from controls at the $p < 0.01$ level.

Chapter 6

Evolving Compound Behavior

In practice, distributed systems exhibit multiple classes of behavior. For example, one such system is the *mobile ad-hoc network* (MANET), which typically comprises a collection of low-power mobile sensors, such as that shown in Figure 2.4(b). MANETs generally include data-driven behaviors related to their task (e.g., distributed temperature sensing), structural behaviors to maintain network connectivity, and temporal behaviors related to coordination and energy management (e.g., synchronizing activity cycles to preserve battery power).

While many techniques have been developed to control such multi-agent systems, e.g., graph-stability [146], switching control strategies [147], adaptive gradient climbing [148], and others [149, 150, 38], the engineering of control strategies for complicated and/or dynamic environments remains a challenging problem [141, 17]. For example, ocean-going MANETs experience highly dynamic conditions, radio wave refraction, as well as numerous seaborne hazards [151, 152]. Moreover, the complexity of such systems also increases exponentially with the number of agents [153]. New techniques that take into account the characteristics of the environment and that are scalable to large multi-agent systems are needed.

Our studies: In this chapter, we investigate the evolution of controllers for a simulated MANET, focusing on a problem that not only requires cooperation among agents to solve, but that includes aspects of all three types of distributed behavior previously discussed. Considering that neural networks are well-suited for control of agents in continuous-time environments, the studies presented here use three related approaches for neuroevolution [78], a technique for evolving artificial neural networks (ANNs), and (for the most part) we focus on genetically homogeneous networks of neural network-controlled agents. In the remainder of this chapter, we describe the different neuroevolutionary techniques we used, present the model problem upon which we conducted our studies, and present the results of two separate studies, one where we evaluate different neuroevolutionary approaches, and another where we focus on the ability of single neuroevolutionary approach to evolve a stable, self-organizing, and scalable solution for our model problem.

6.1 Methods

This section describes the model problem we addressed and the three neuroevolutionary systems we used for evolving distributed behavior in MANETs.

In the broader context of multi-agent systems, the taxonomy provided by Panait and Luke [153] places this study in the *team learning* category, where a single evolving individual learns the behavior for the entire group. Moreover, the agents in this study engage in *direct communication* through message-passing, and no mechanism for indirect communication (e.g., stigmergy) is provided. Finally, the scenario that we have selected for study is related to *cooperative target observation*, where a group of agents is tasked with collective observation of a target. The specifics of this scenario will be explained in more detail below. Dorigo

et al. [154] and Baldasarre, Parisi, and Nolfi [155] have also studied coordinated behavior of robotic swarms where individuals were able to physically attach to one another. In contrast to those studies, which focus on aggregation and coordinated motion of connected agents, our study focuses on controlling independent agents for a task that requires coordination among distant agents.

A closely related study is that by Hauert, Zufferey, and Floreano [156], where a neural network controller for unmanned aerial vehicles (UAVs) was evolved via a genetic algorithm. Here, the UAVs were to establish and maintain a multi-hop communications link between a base station and a target without global or agent-relative positioning information. The key differences between this study and our own are that we include explicit sensory information related to the relative positioning of agents and have focused on a coverage-related problem. Finally, D'Ambrosio et al. [157] investigated the use of neuroevolution on a coverage-based problem where agents were able to sense the boundaries of their environment, but were not able to sense other agents. In contrast, we study agents that are able to sense their relative positions in an environment that does not place limitations on movement.

6.1.1 Grid-Coverage Problem

Figure 6.1 is an illustration of a mobile network for oceanic monitoring. Here, agents in the network must not only coordinate their movement in order to remain connected via wireless links, but they must also spread out to maximize the coverage area of their sensors. As the first step towards evolving solutions to such problems, we have defined the *grid-coverage problem*, where a mobile network of autonomous agents is to be maneuvered into a grid-like formation within a virtual physical environment, all while maintaining network connectivity.

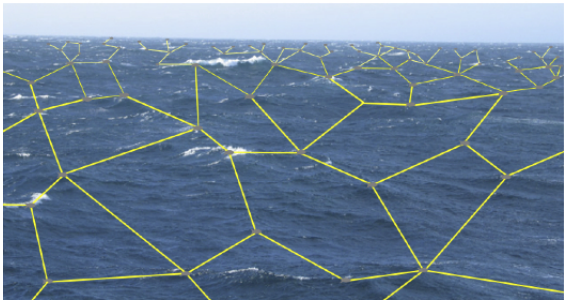
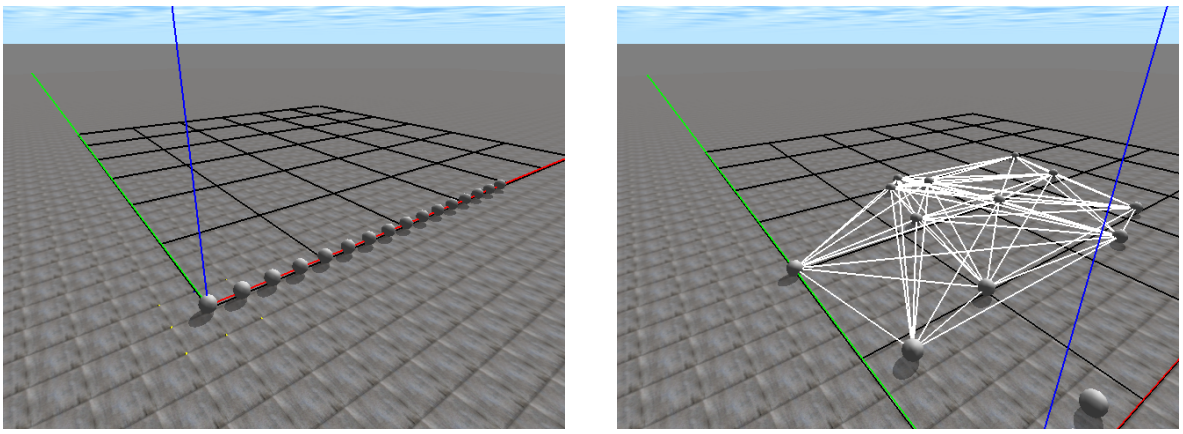


Figure 6.1: Illustration of a network of mobile aquatic sensors for oceanic monitoring. Such a network could be used for monitoring oil spills and studies of ocean life.

General solutions to coverage problems such as this require coordination among agents. For example, in a mobile sensor network, only a limited number of nodes may have access to a base station. In this case, disconnectedness can have disastrous consequences on the effectiveness of the MANET. We note that the environment used for the grid-coverage problem can easily be made more challenging, e.g., by randomly introducing or removing agents, altering the shape and location of the grid (even during the simulation), or by using a more complex environment that includes three-dimensional features, such as waves, hills, and mountains.

Figure 6.2(a) depicts the environment and initial configuration for a network whose behavior will be evolved via neuroevolution. As shown here, a 6×6 grid of cells, each $4m \times 4$m in size, is arrayed in a flat environment. Each of 16 mobile agents is modeled as rolling sphere. The physics-based characteristics of this environment are calculated by the Open Dynamics Engine (ODE, version 0.11.1, http://www.ode.org). Within this environment, agents have a variety of sensors and effectors, summarized in Table 6.1. Figure 6.2(b) depicts this same network following 4s of control by an evolved neural network. Shown here are the communication links connecting agents in this network. Specifically, all agents that are within 10m of each other are able to communicate. When an agent transmits (by raising



(a) (b)

Figure 6.2: Starting configuration of agents (a) and position of agents 4s into a simulation (b). White lines represent connections between agents; agents can transmit data only to those other agents to which they are connected. The grid agents are to cover is outlined with black lines; x, y, and z axis represented by red, green, and blue lines, respectively.

Table 6.1: Sensors and effectors of individual agents for the grid-coverage problem.

Name	Sensor/effector	Description
rx0...rx3	sensor	directional radio strength; each sensor covers a 90° "pie slice" around the agent
v0...v2	sensor	velocity vector (x, y, z)
p0...p2	sensor	absolute position (x, y, z)
f0...f2	effector	force vector (x, y, z)
tx	effector	transmit message

its tx effector above 0), the appropriate rx sensor of all agents within 10m is set to the inverse of the distance from sender to receiver. The rx sensor used for a given agent is based on the orientation of the agents in space; each rx sensor covers a 90° "pie slice" around the agent. When transmissions are received, the strongest (nearest agent) is used.

Evaluating the performance $G(M, n)$ of a MANET on the grid-coverage problem is a straightforward process. Specifically:

$$G(M, n) = \frac{cells(M)}{\min(n, |M|)} \tag{6.1}$$

where M is the MANET being evaluated, $cells(\cdot)$ calculates the number of unique cells in the grid that are occupied by at least one agent in M, and n is the number of cells present in the grid. In all of the studies presented here, $n = 36$ (the grid is always 6×6).

6.1.2 Neuroevolution

For this study into distributed control, we used three different neuroevolutionary systems, NEAT [78], HyperNEAT [53], and Multi-agent HyperNEAT [158], depicted in Figure 6.3. Each of these systems was used to evolve ANNs which were used as controllers for both the movement and communication behavior of agents within a mobile network.

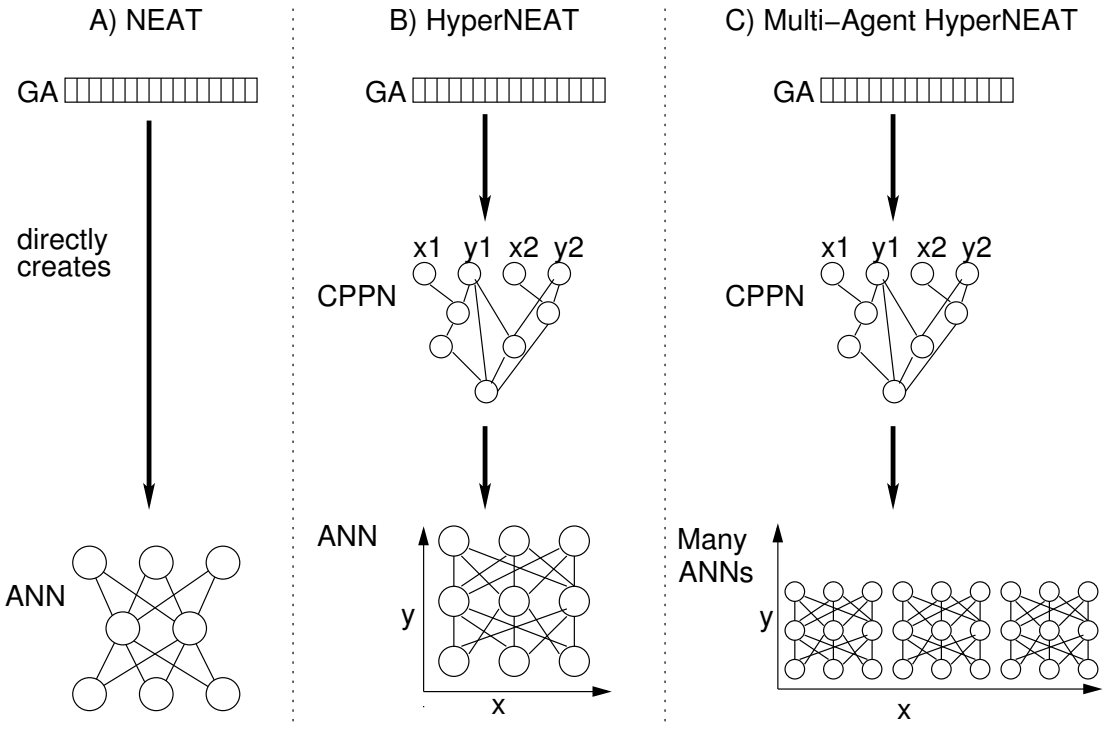


Figure 6.3: A graphical depiction of how NEAT, HyperNEAT, and Multi-agent HyperNEAT produce ANNs.

NEAT. The first neuroevolutionary approach we used was NeuroEvolution of Augmenting Topologies (NEAT) [78]. Illustrated in Figure 6.4, NEAT is fundamentally a genetic

algorithm that uses a *direct encoding*, where each gene encodes either a neuron or link in an artificial neural network. A key advantage of NEAT compared to other neuroevolutionary approaches is that it is able to evolve the structure of the neural network. Specifically, NEAT starts with ANNs that include neither a hidden layer nor recurrent connections. Over time, using a process called *complexification*, NEAT is able to evolve genomes that include both hidden neurons and recurrence. In this way, NEAT does not bias or constrain the solution to any particular network topology.

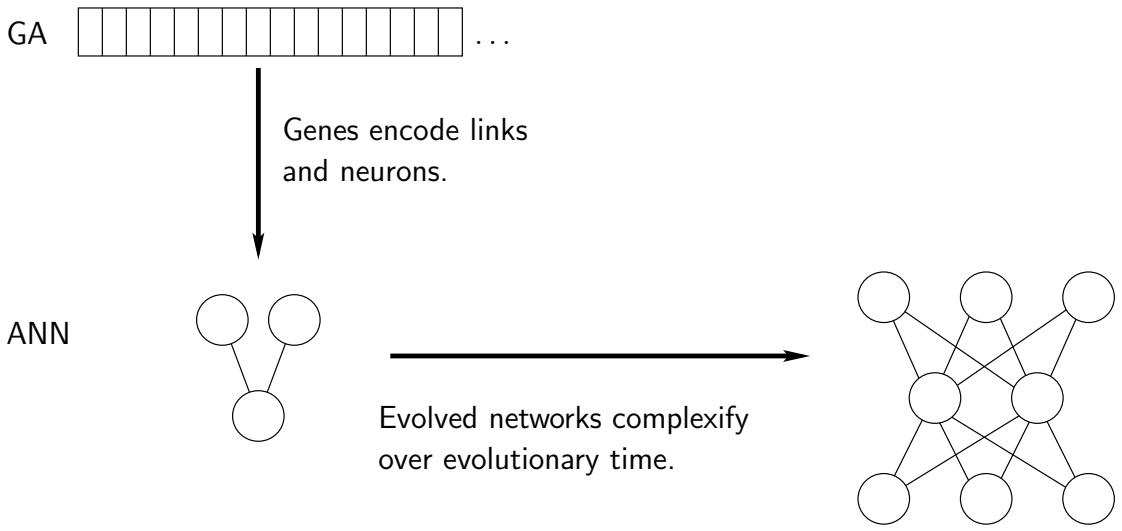


Figure 6.4: Illustration of NEAT, the neuroevolutionary approach used in this study.

HyperNEAT. Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [53] is a genetic algorithm that uses an *indirect encoding*, where each gene encodes a specific piece of a Compositional Pattern Producing Network (CPPN); the CPPN, once fully constructed, produces the ANN (Figure 6.3b). Within HyperNEAT, a CPPN is effectively a complex mathematical function that takes as input the coordinates of a source neuron and target neuron and produces as output the weight for the link connecting them. By querying each possible source and target neuron, an ANN is produced from a CPPN. The assignment of

coordinates to neurons in HyperNEAT is known as the *substrate structure*, and is a key part of using HyperNEAT, as will be shown in Section 6.2. For evolutionary purposes, CPPNs are represented as graphs, where the nodes are mathematical functions such as gaussians, sigmoids, and sines, while the edges govern the composition relationship among the functions. The mathematical functions are capable of producing any ANN. HyperNEAT is an extension of NEAT that uses the NEAT algorithm to evolve CPPNs that produce ANNs, rather than directly evolving ANNs.

The primary advantage to HyperNEAT is that its use of an indirect encoding enables the creation of ANNs that exhibit symmetry and repeated motifs. Thus, HyperNEAT can capitalize on the regularity present in many domains and evolve solutions to more complex problems [159]. However, HyperNEAT has been shown to have difficulty making minor exceptions to regular patterns [160].

Multi-agent HyperNEAT. The third approach that we used was Multi-agent HyperNEAT [158], which is an extension of HyperNEAT that enables one genome to encode a set of ANNs that represent controllers for a heterogeneous team (Figure 6.3c). Specifically, Multi-agent HyperNEAT adds an additional input to the CPPN that represents which ANN is being evolved. The CPPN is then used to produce one ANN for each agent. As a result of the additional input, Multi-agent HyperNEAT is able to evolve ANNs that make use of many common strategy elements, but may also differ from one another [158]. We note that a MANET constructed with Multi-agent HyperNEAT is genetically homogeneous, even though the neural networks controlling each agent may vary.

6.2 Evaluation of Neuroevolutionary Approaches

In this section, we present our evaluation of NEAT, HyperNEAT, and Multi-agent HyperNEAT on the grid-coverage problem. For all of the experiments described here, we used the fitness function described in Section 6.1.1. Each agent had the sensors and effectors summarized in Table 6.1, and we conducted 30 trials of each treatment. Each treatment was executed for 200 generations on a population of size 100. Mutation rates for adding links and nodes to NEAT were set to 0.05 and 0.03, respectively. Mutation rates for adding links and nodes to the CPPN for HyperNEAT-based treatments were set equivalently. We note that this approach may allow differences in *effective* mutation rates between NEAT and HyperNEAT-based treatments.

6.2.1 Evolving Mobile Sensor Controllers

In our first experiment, we used a 16-node swarm and simple substrate structures for the HyperNEAT and Multi-agent HyperNEAT treatments. Specifically, we used a fixed topology that contained one hidden layer, where each node in the input layer was linked to each node in the hidden layer, which was in turn linked to each node in the output layer. NEAT, of course, evolves not only the topology of the ANN, but also connection weights, so no additional structures were required.

Figure 6.5 plots the performance, the fraction of optimal fitness, achieved by NEAT (N), HyperNEAT (H1), and Multi-agent HyperNEAT (M1), averaged across all 30 trials. All three techniques achieved between 60% to 70% of maximum fitness before reaching a plateau. While treatment N had the overall best performance, tests for statistical significance revealed that treatment N differed from treatment H1, but no other pairs were significantly different

from each other (*Kruskal-Wallis* multiple comparison, $p < 0.01$).

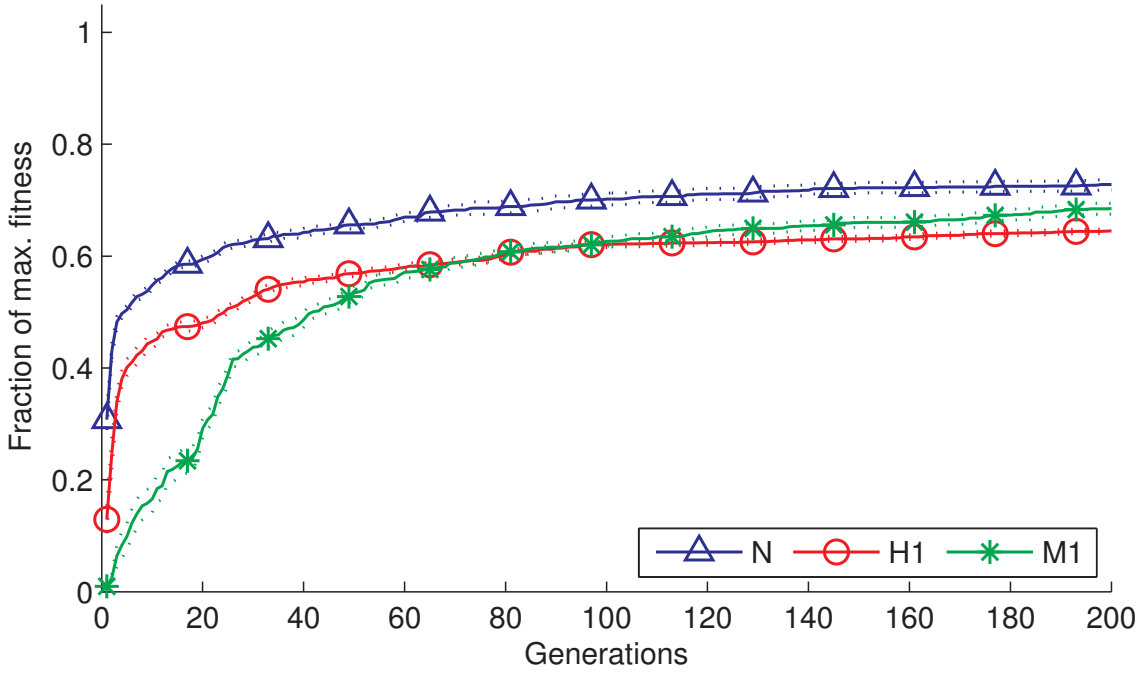


Figure 6.5: Fraction of maximum fitness achieved with NEAT (N), HyperNEAT (H1), and Multi-agent HyperNEAT (M1) on networks comprising 16 agents.

Previous research has demonstrated that the structure of the underlying substrate used by HyperNEAT affects its ability to effectively solve problems [161]. Selecting an inferior substrate can result in substandard performance. To ensure that the performance of HyperNEAT-based systems was not unfairly affected by the selection of a poor substrate structure, we next tested several additional configurations, all of which are described in Table 6.2.

Figure 6.6 plots the fraction of maximum fitness of all NEAT- and HyperNEAT-based treatments. Each box represents a single treatment, and was constructed from the maximum fitness achieved by the best performing individual from each of the 30 trials. Treatments are positioned in ascending mean rank order from left to right. In general, the mean performance ranges from 60% to 70% of maximum fitness. Overall, while the maximum performance

Table 6.2: Substrate structures for HyperNEAT and Multi-agent HyperNEAT treatments used in this paper. Treatments H1-H4, D1-D5, F1, and T used HyperNEAT, while treatments R1-R2, M1-M3, and F2 used Multi-agent HyperNEAT.

Treatment	Description
H1-H4	Each layer of the neural network is given its own 2-D coordinate frame. Inputs to the CPPN for each neuron are a triple (x, y, l), where x, y are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
D1-D5	Each layer of the neural network is given its own 3-D coordinate frame. Inputs to the CPPN are a tuple (x, y, z, l), where x, y, z are the coordinates of the neuron, and l is the neural network layer. Each treatment uses a different substrate structure.
M1-M3	Each neuron is described with a tuple (x, y, l, i), where x, y are the coordinates of that neuron, l is the neural network layer, and i is a unique identifier based on the number of agents (e.g., for a 16 agent network, i ranges from $0 \ldots 15$). M1 uses the substrate structure from H1, M2 uses the substrate structure from D1, and M3 adds a bias neuron to M1.
R1-R2	Each neuron is described with a tuple (x_g, y_g, x_r, y_r, l), where x_g, y_g are the global coordinates for that neuron, and x_r, y_r are relative coordinates for that neuron, and l is the neural network layer. Relative coordinates were calculated by scaling x_g, y_g by i/m, where i is a unique identifier for that agent, and m is the total number of agents. R1 uses the substrate structure from H1, while R2 uses the substrate structure from D2.
F1-F2	Each neuron is described with a pair (x, y), and all layers are placed on the same coordinate frame. Each treatment uses a different substrate structure.
T	CPPN is modified to return two weights, one for each link between layers; uses the same substrate structure as H1.

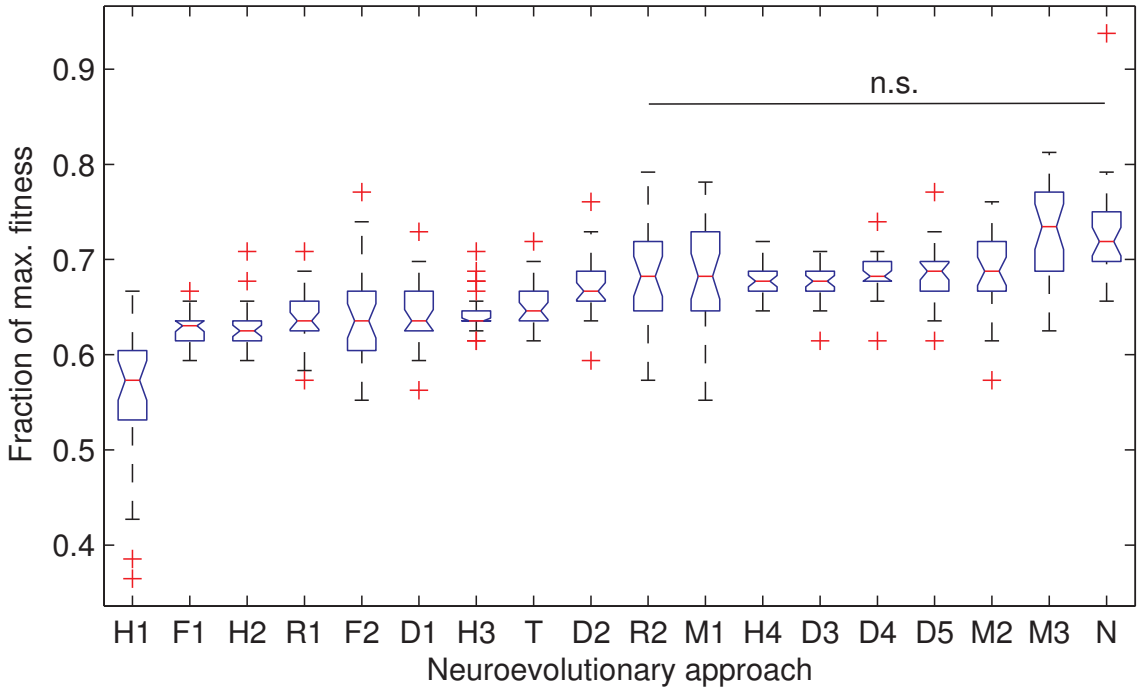


Figure 6.6: Fraction of maximum fitness achieved with NEAT, HyperNEAT, and Multi-agent HyperNEAT on networks comprising 16 agents. Multiple different substrate configurations were tested for HyperNEAT and Multi-agent HyperNEAT treatments. Approaches are in ascending mean rank order from left to right. Treatment N (far right) is the NEAT-based treatment; see Table 6.2 for descriptions of HyperNEAT-based treatments.

of NEAT exceeded that of all HyperNEAT-based treatments, no statistically significant differences between the highest performing treatments were found ($p < 0.01$; the "n.s." annotation in Figure 6.6 indicates these treatments). Of the HyperNEAT-based treatments, the only one whose mean performance (though not mean rank) exceeded that of NEAT (treatment N) was M3, which employed Multi-agent HyperNEAT. Many of these treatments were also tested with an additional (second) hidden layer; results were not statistically significantly different from a single hidden layer (data not shown).

Table 6.3: Potential factors that affect the grid-computing problem difficulty. The name of factors that have a more than minimal affect on results are listed in bold text.

Name	Description of experimental test
num-agent	Varied the number of agents included in the simulation.
sensors	Varied whether or not agents had access to sensor data.
radio	Varied the range of the radio to see if the range compared to cell size mattered.
velocity	Varied the velocity at which a node was considered as part of the fitness calculation.
id	Provided each agent with an identifier to enable behavioral differentiation.
grid-size	Varied the number of cells in the grid to see if performance was an artifact of percentage of squares filled.

6.2.2 Key Contributors to Problem Difficulty

Our next set of experiments investigated some of the factors that contributed to the difficulty of the grid-coverage problem. In addition to the NEAT-based treatment (N), we selected two variants of HyperNEAT (D2) and Multi-agent HyperNEAT (M1) for further analysis. Here, we present the results of two of these investigations, specifically regarding the availability of sensor data and the number of agents included in the simulation.

We investigated the effect of six factors on these three treatments, summarized in Table 6.3. Two factors, the number of agents included in the simulation and the presence of sensor data, significantly impacted the performance of the three approaches. The other factors produced results that were frequently statistically significant, but did not have a qualitatively large impact on overall performance. In this section, we focus on the two primary factors that affected problem difficulty.

The first factor that impacted performance was the availability of sensor data. For these treatments, we zeroed out the relevant sensors and re-ran each treatment to produce new ANNs. Specifically, we ran one set of experiments where we removed the radio strength

sensing capability and one where we removed the location sensing capability. We note that these results do not reveal whether the previously evolved ANNs used this information, but rather can be used to understand what sensor information is required to find general solutions to the grid-coverage problem.

Figure 6.7 and 6.8 depict the results of removing the radio sensing and location sensing capabilities, respectively, from all three approaches. The performance of HyperNEAT and Multi-agent HyperNEAT dropped significantly (up to 20%) compared to their performance in Figure 6.6, indicating that these sensors were most likely a key part of their previous solutions. The performance of NEAT was less affected by the loss of these sensors, indicating that the strategies evolved by NEAT were less dependent on their environment. In both cases, the performance of NEAT is statistically significantly better than the HyperNEAT-based treatments ($p < 0.01$), while the two HyperNEAT-based treatments were not significantly different from each other at the $p < 0.01$ level.

While it is well-known that the number of agents being controlled is a significant factor for multi-agent systems [153], it has been conjectured that generative and indirect encodings might be better equipped to deal with this challenge than direct encodings [158]. To better understand how the number of agents might affect performance on the grid-coverage problem, we performed a series of additional treatments that varied the number of agents from 2 to 32. Figure 6.9, 6.10, and 6.11 depict the performance of NEAT, HyperNEAT, and Multi-agent HyperNEAT, respectively, as the number of agents is varied. The same pattern of performance is repeated across all three treatments, where we see that as the number of agents increases, overall performance steadily decreases. We note that in these figures, the performance of a given treatment is relative to the number of agents. Thus, while

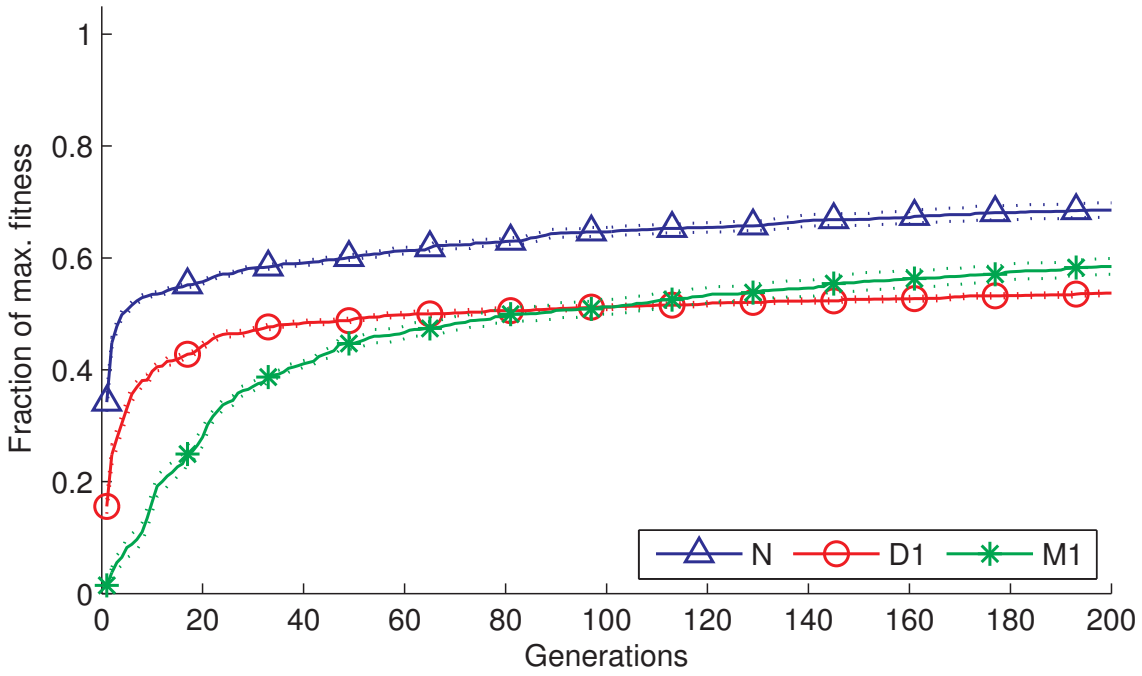


Figure 6.7: Performance of NEAT (N), HyperNEAT (D1), and Multi-agent HyperNEAT (M1) decrease when radio sensors are removed.

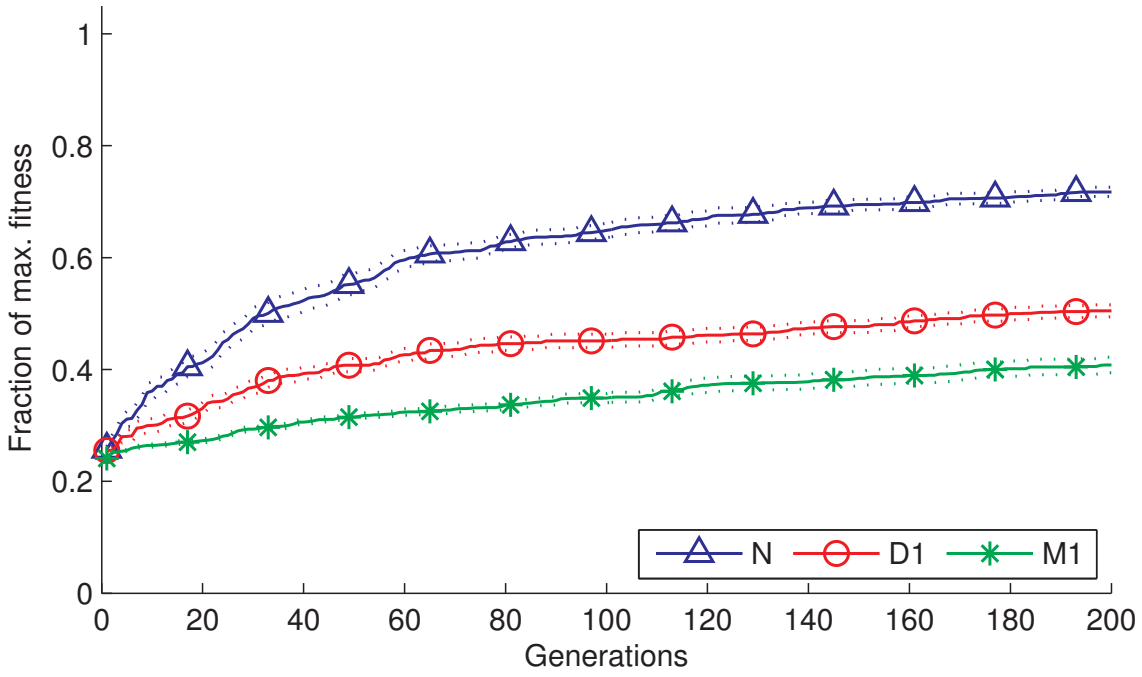


Figure 6.8: Performance of NEAT (N), HyperNEAT (D1), and Multi-agent HyperNEAT (M1) decrease when location sensors are removed.

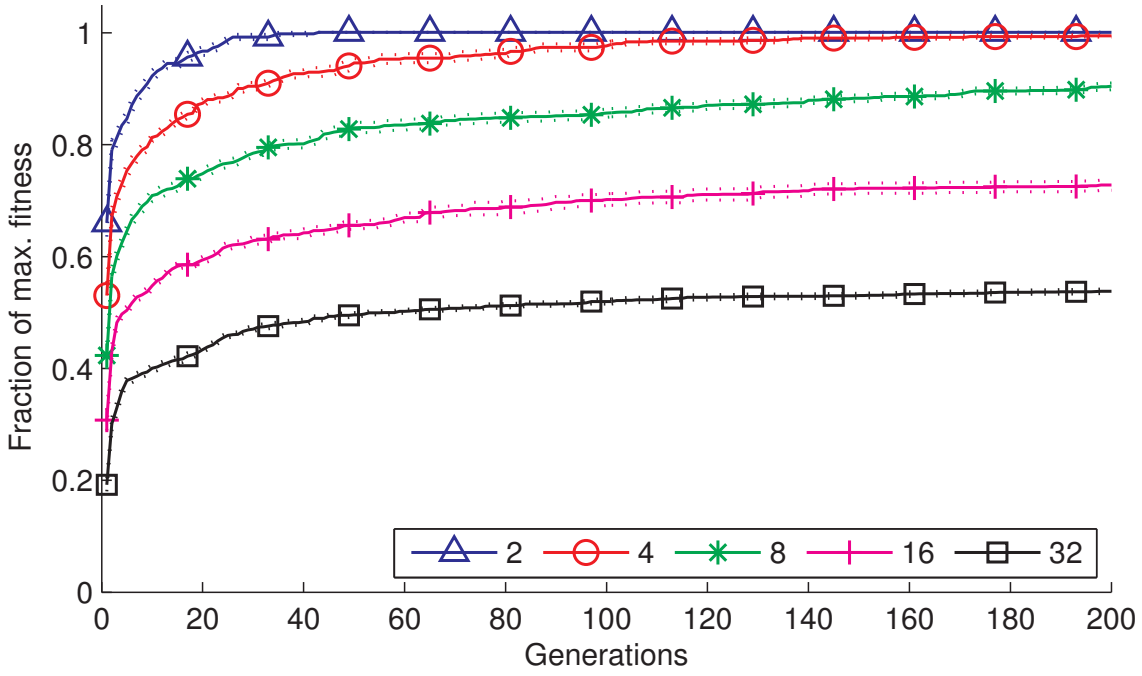


Figure 6.9: Performance of NEAT for different numbers of agents.

performance (measured as the fraction of maximum fitness) decreases as the number of agents is increased, the total area covered by the agents increases. Interestingly, the performance of the NEAT-based treatment degraded somewhat more slowly than the HyperNEAT-based treatments (mean performance drop between network sizes 11.5% for NEAT, 13.3% for HyperNEAT, $p < 0.01$), implying that NEAT may be more effective at evolving scalable control strategies, which we investigate in the next section.

6.2.3 Evidence for Self-Organization

A key component of biological self-organization is whether agents react to each others' behavior. In most of the experiments conducted, agents had the capability to modulate their communication behavior. Specifically, when the tx output was greater than 0.0, an agent broadcast that signal to its neighbors. A measure of the strength of that signal ($1/distance$)

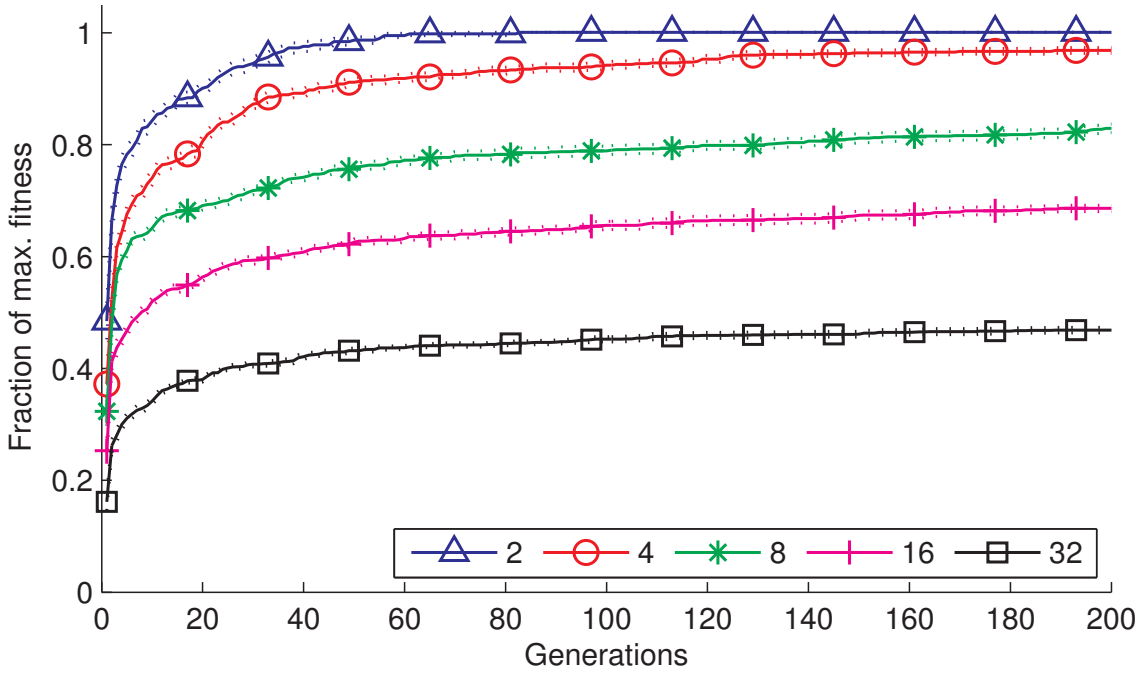


Figure 6.10: Performance of HyperNEAT for different numbers of agents.

was then applied to the appropriate rx input. In our experiments, we found a significant fitness difference between treatments that included this capability versus those that did not. However, this does not address the question of whether the treatments that included signal strength were actually using it. To test whether or not the controllers produced via neuroevolution were in fact engaging in self-organizing behaviors, we took the final solutions from each of NEAT, HyperNEAT, and Multi-agent HyperNEAT treatments and tested them in an environment where all rx inputs were set to 0.

Figure 6.12 plots the original fitness of each dominant vs. its fitness with their rx inputs set to 0. Points that are above the diagonal (all but 2) represent solutions whose fitness is greater when sensors are enabled. For example, the point near $(0.5, 0.7)$ represents a single trial whose fitness with sensors enabled (the conditions under which it evolved) reached 70% of maximum fitness, while when its sensors are disabled (set to 0), it achieved only 50% of

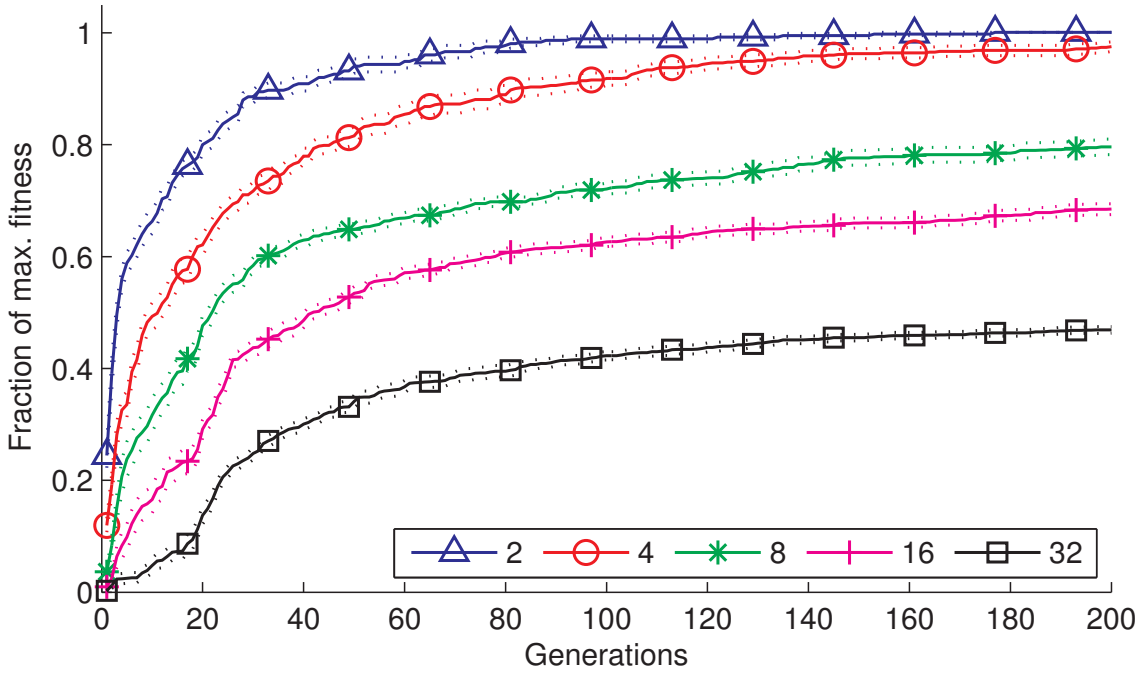


Figure 6.11: Performance of Multi-agent HyperNEAT for different numbers of agents.

its maximum fitness. That all but two trials (both NEAT-based) had greater fitness with sensors than without is evidence of self-organization.

In the next section, we focus on a single neuroevolutionary system, NEAT, and explore the questions surrounding self-organization and scalability on the grid-coverage problem in more detail.

6.3 Evolving Scalable Behavior

In this section, we present our experiments and results on using NEAT to evolve neural network controllers for the nodes in a mobile ad hoc network. Here, we specifically focus on evolving stable, self-organizing, and scalable behaviors.

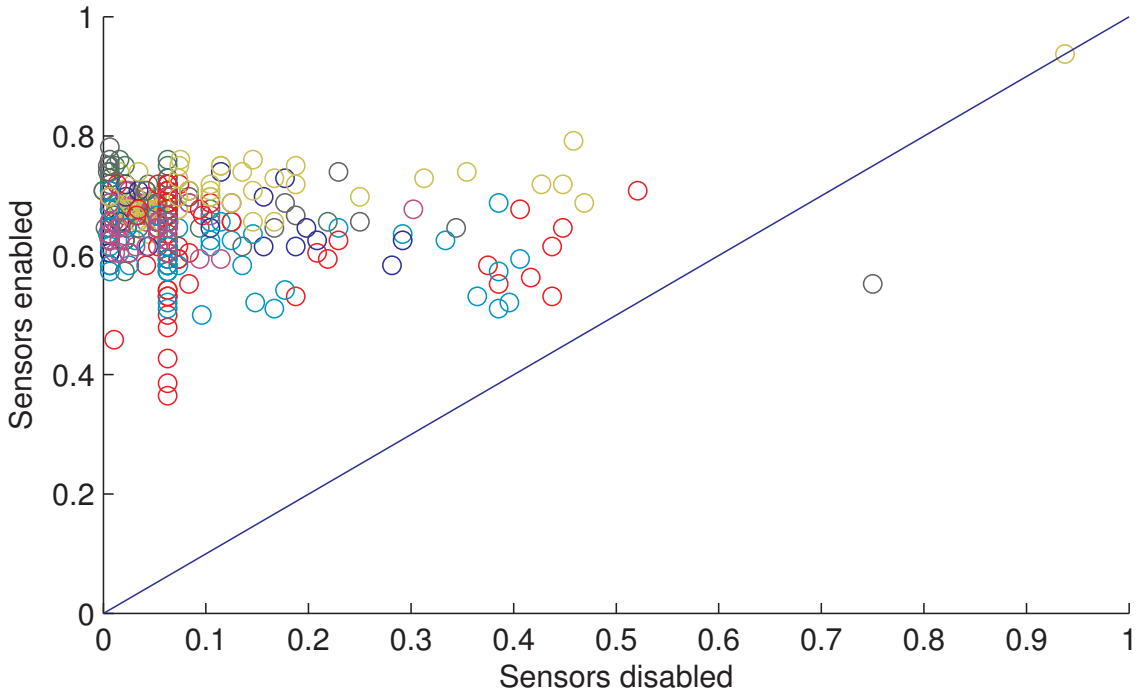


Figure 6.12: Fraction of maximum fitness with sensors enabled vs. sensors disabled. Each point in this plot represents a single trial; all treatments from Figure 6.6 are represented. Fitness is depressed when sensors are disabled, indicating that evolved solutions are self-organizing.

6.3.1 Motivating experiment

Our first experiment was a straightforward attempt at evolving neural networks that could solve the grid-coverage problem, and it provides the motivation for the following experiments. Specifically, we first examine a fitness function that was based only on the performance of the MANET on the grid-coverage problem. The specific fitness function used here was simply that shown in Equation 6.1, which rewarded individuals for the fraction of cells that were occupied by nodes in the MANET at the time of fitness evaluation. We configured the simulation to run for 10s, and calculated fitness at the end of the simulation.

Figure 6.13 plots the mean normalized fitness of the dominant (most fit) individual per generation across 30 separate trials for both 8-node ($FF8$) and 16-node ($FF16$) networks (error bars in this figure and all following are 95% confidence intervals constructed via 200-fold bootstrapping). We define *normalized fitness* as f/f_{max}, where f_{max} is the maximum possible fitness that is achievable given the fitness function and network size. In this treatment, fitness and normalized fitness are identical, although that is not the case in many of the following experiments. In Figure 6.13, we see that the dominant individuals in 8-node networks reach their optimal fitness after approximately 40 generations, while 16-node networks averaged only $71.5 \pm 0.6\%$ after 200 generations.

Upon examination of the evolved behaviors, the reason for the decline in fitness of 16-node networks became apparent: For both 8- and 16-node networks, the evolved strategies were primarily stochastic, and relied upon the continual movement of nodes throughout the environment. One such behavior is depicted in Figure 6.14, which shows the position and velocity of each node in the 16-node network over 20s of simulation time. In this figure, nodes start in an array along the $x-$axis, and generally move in a spiral pattern within the

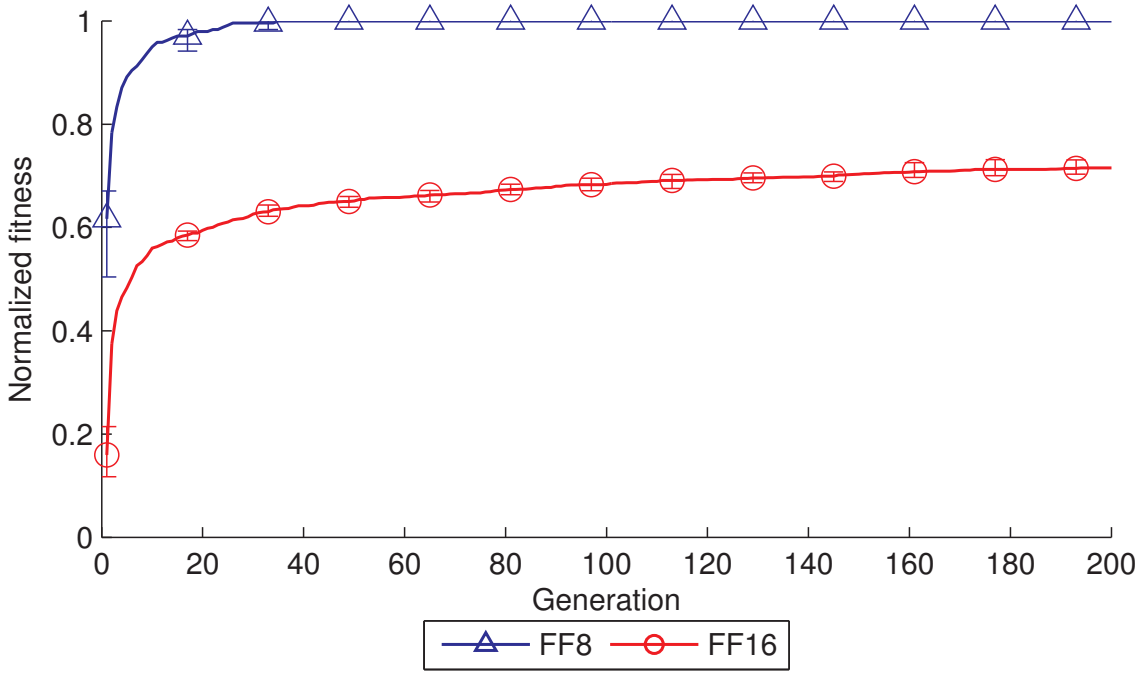


Figure 6.13: Normalized fitness of 8- and 16-node fixed-size networks using a fitness function that rewards only for grid-coverage.

region bounded by the grid. This behavior was also prevalent in the smaller 8-node networks. The reason this strategy is effective is related to the size of the grid relative to the number of nodes in the network. On average, if the grid is large relative to the size of the network, and nodes maintain a small distance between each other while moving over the grid, nodes are likely to be in different cells at the time of fitness evaluation. This avoidance strategy is not unlike that exhibited in flocks of starlings [11], where individuals adjust their movements to maintain a "safe" distance amongst themselves.

However, from the perspective of engineering controllers for a MANET, these evolved behaviors are lacking in two significant ways: First, the evolved behaviors are *unstable*. We expected to see behaviors where nodes would deterministically move to unique cells in the grid and stop moving. Instead, the nodes moved continually in a spiral pattern, which would waste energy in a real network. Second, these behaviors do not *scale*. We hoped to evolve

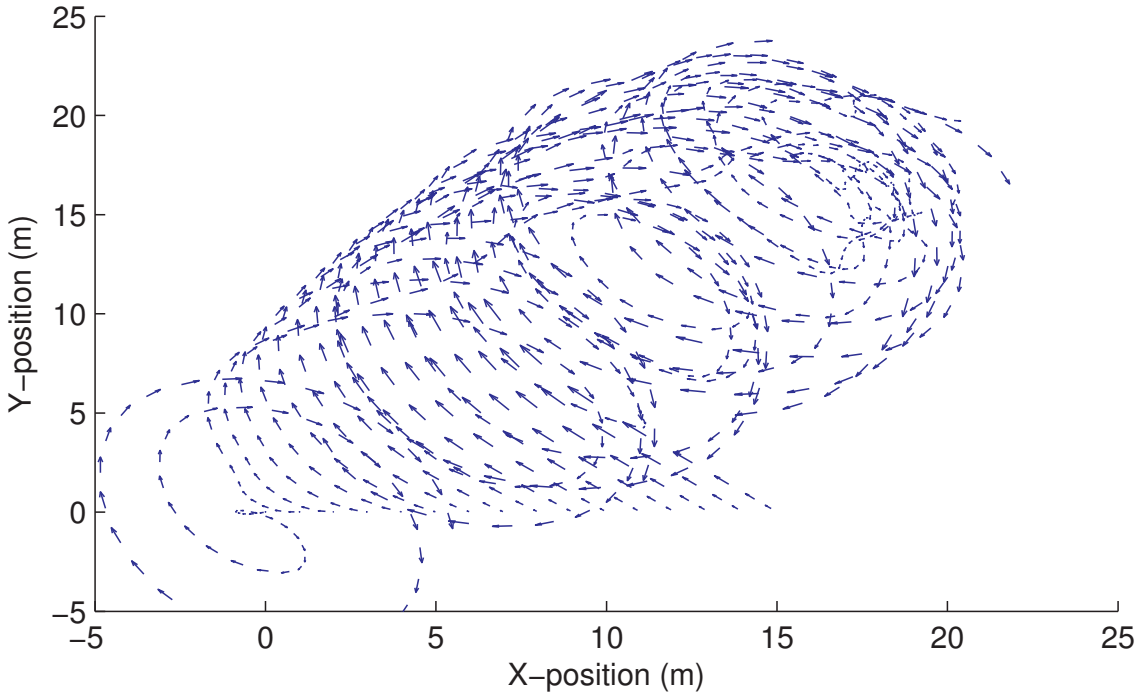


Figure 6.14: Sample velocity and position of a 16-node network following 20s of control ($FF16$ treatment). Nodes start at $(x, 0)$, and then move in a spiral for the remainder of the simulation.

effective behaviors for networks of varying size. However, we found that normalized fitness decreased as network size increased, a result which we verified on larger 32-node networks and both larger and smaller grids. In all cases, the key factor was the number of nodes being controlled, which is known to be a challenge for multi-agent systems [153]. In the following experiments, we examine ways to improve network stability and scalability.

6.3.2 Stable networks

Based on the behaviors observed in the previous experiment, we first explored ways to improve network stability. The goal here was to discover networks whose nodes would move to and then settle in a location, instead of continuing to move throughout the environment. Our first approach was to devise a fitness function that included a speed component, such

that networks with nodes that were moving slowly at the time of fitness evaluation would have a higher fitness than a network whose nodes were moving quickly. This fitness function, $SF(M)$, rewarded for a reduction in the average speed at which nodes in M (the MANET) were moving, specifically:

$$SF(M) = \frac{G(M, 36)}{10 \cdot \max(0.1, \overline{\text{spd}(M)})} \tag{6.2}$$

where $G(M, 36)$ is performance on the 36-cell grid-coverage problem, and $\overline{\text{spd}(M)}$ is the average speed of all nodes in M at the end of the simulation. The other components of this fitness function, scaling by 10 and $\max(\cdot)$, were used to ensure that the resulting fitness score is in the range [0..1]. This fitness function thus rewards MANETs that quickly move to, and settle, in a stable configuration.

Figure 6.15 plots the mean normalized fitness of the dominant individual per generation over 30 trials on networks of 16 nodes. The mean normalized fitness of dominant individuals under this treatment is $55.3 \pm 1.2\%$ after 200 generations, which is still quite low.

Upon examination of the evolved behaviors, we did indeed find that the networks exhibited the desired behavior, where nodes stabilized into a fixed position after reaching their target cells. An example of this behavior is shown in Figure 6.16, which shows the position and velocity of each node during 20s of simulation time. Nodes again start in an array along the $x-$axis, and slowly move to different grid cells, though many nodes share cells.

One key difference we noticed between this behavior and that shown earlier in Figure 6.14 is that here the behavior of individual nodes is based on their starting location in the environment, and does not appear to be stochastic. *Information entropy* is frequently used to characterize stochastic processes [162]. To measure information entropy, $H(X)$, of a mobile

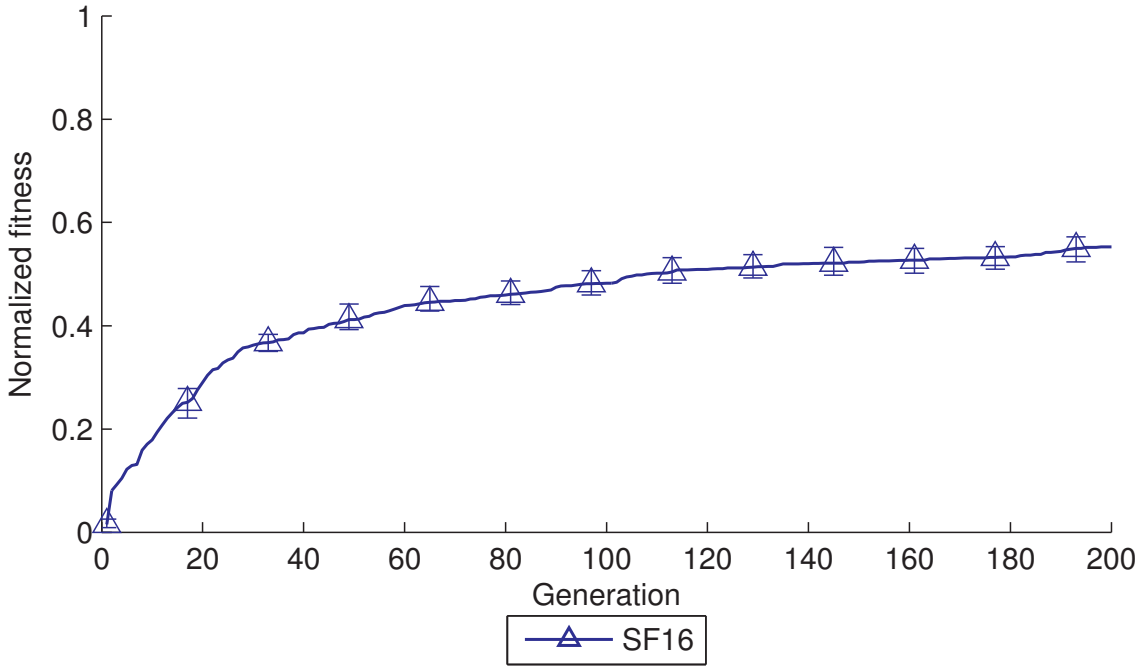


Figure 6.15: Normalized fitness for 16-node networks that are rewarded for reduced speed.

network, we leverage the following definition:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{6.3}$$

where $X = \{x_1, \ldots, x_n\}$ is the discrete random variable representing the various states of the network and $p(\cdot)$ is the probability mass function of X. We define the states of X based on the cells of the grid that are occupied by nodes in the network. Specifically, each possible value of X is a length-k bitstring, where k is the number of cells in the grid (36, in this study). A 1 at position i in this bitstring then represents that cell i is occupied by at least one node, while a 0 indicates that cell i is not occupied. The frequencies of these states during a simulation defines $p(\cdot)$. Intuitively, a high value for information entropy represents a network that exhibits many states with low frequency, while a low value for information entropy represents a network that exhibits few states with high frequency. We

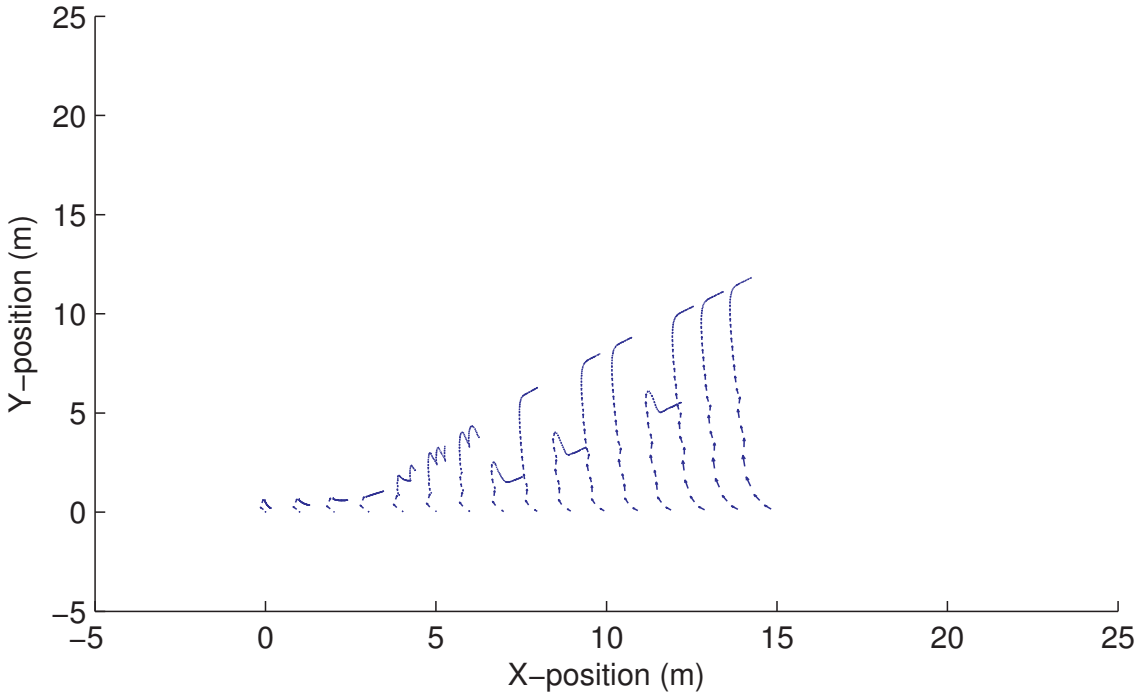


Figure 6.16: Velocity and position of 16 nodes in the network evolved under treatment $SF16$, which rewards for a reduction in the average speed of nodes at the end of the simulation.

assume that low information entropy is a desirable property of engineered systems. For brevity, throughout the remainder of this paper we will simply refer to this calculation as the entropy of a network.

Figure 6.17 plots the mean entropy for the dominant individuals per generation over all 30 trials for two treatments: $FF16$, described in Section 6.3.1, whose fitness function did not include a velocity component; and $SF16$, described in this section, where a reward for reduced velocity was included as part of fitness. As would be expected, here we see that entropy of the $SF16$ treatment is significantly less than that of the $FF16$ treatment, indicating a more stable behaviors.

To further explore the relationship between entropy and behavior, we also devised a fitness function that rewarded explicitly for a reduction in entropy, with no consideration

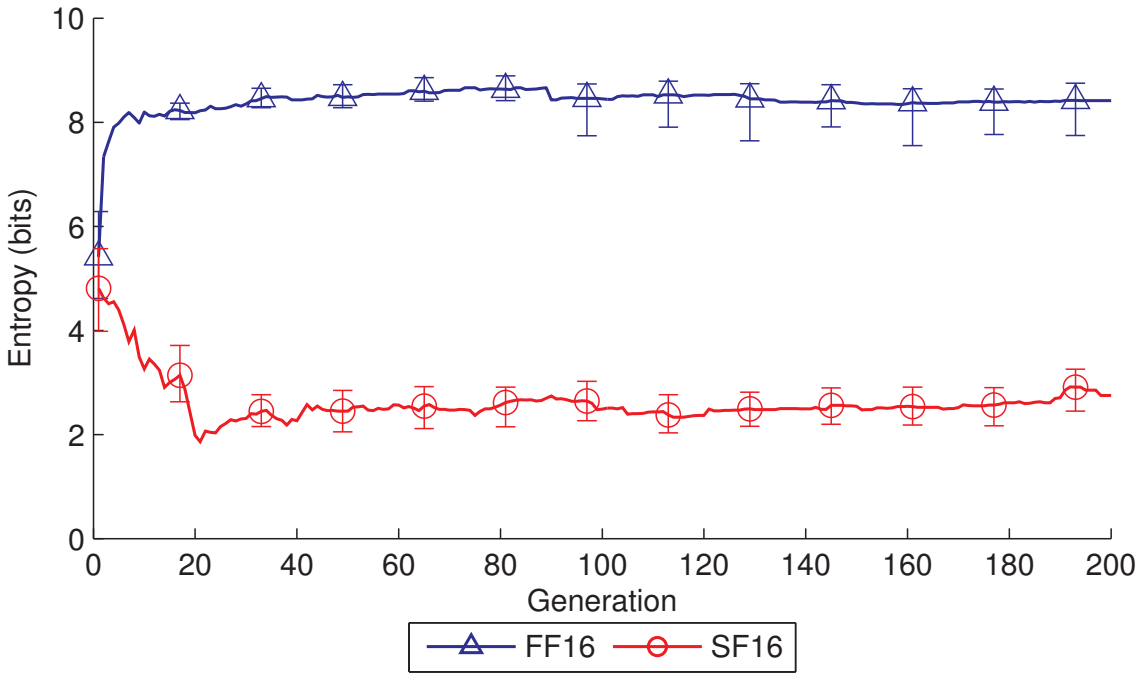


Figure 6.17: Entropy on 16-node networks under a treatment rewarding only for grid-coverage ($FF16$) and the grid-coverage plus reduced velocity ($SF16$).

given to speed. Though their performance on the grid-coverage problem was similar, the evolved behaviors were markedly different. Specifically, when using a fitness function that rewarded for reduced entropy instead of average speed, fitness was in fact declining near the end of the simulation. An example of this behavior is depicted in Figure 6.18, where nodes start by moving very slowly (short arrows) but then accelerate as the simulation progresses (long arrows) eventually moving off the grid. For this reason, while we consider low entropy to be a desirable characteristic, our subsequent experiments include speed-reduction as a component of fitness instead of an explicit reward for reduced entropy.

While we found that rewarding for reduced speed improved both stability and entropy of the evolved behaviors, we also noticed that these behaviors were *static*; that is, individual nodes no longer reacted to the presence (or absence) of neighboring nodes.

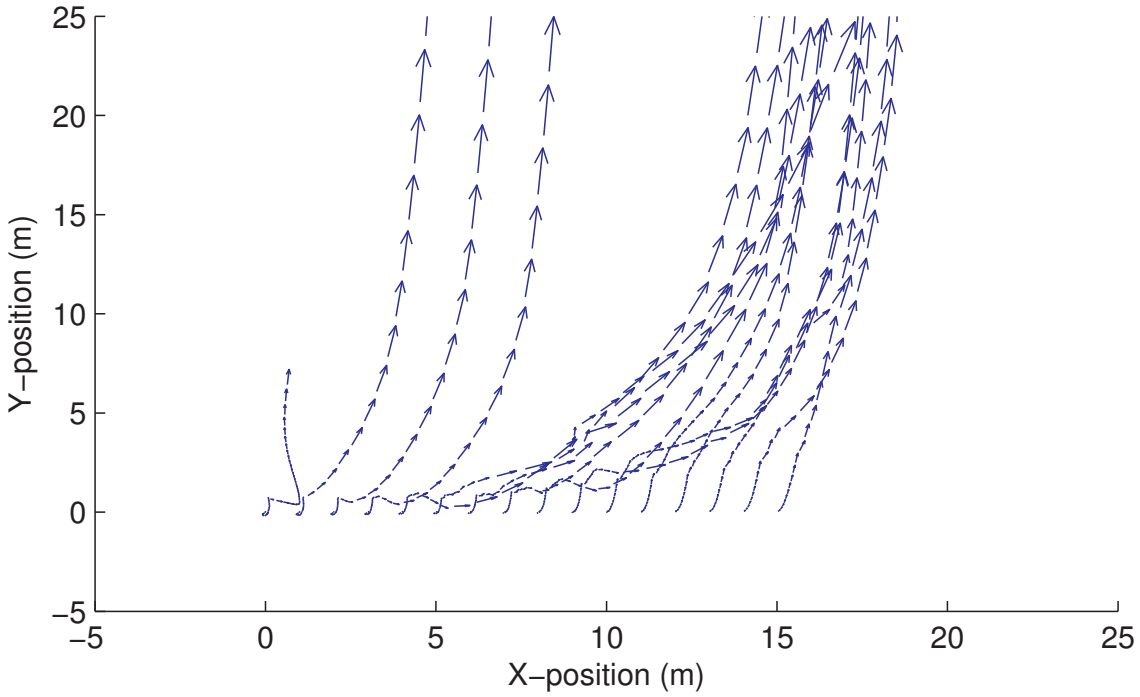


Figure 6.18: Velocity and position of 16 nodes in a network evolved a treatment that rewarded for a reduced entropy explicitly; nodes did not stabilize into a fixed position, instead moving off the grid.

6.3.3 Reactive Networks

In the previous sections, we have looked at a variety of approaches related to the evolution of controllers for MANETs, and identified three main challenges. The first, which we have addressed by rewarding for a reduction in average node speed, was that evolved behaviors tended to be stochastic. Second, we found that when fitness includes a reward for stability, the evolved behaviors were not reactive. Finally, we found that fitness decreased as the number of nodes in the network increased, indicating scalability as a concern as well. To address these final two challenges, we investigated two new approaches that varied the number of nodes in the network during the course of the simulation. Our hypothesis was that if the number of nodes in the simulation changed over time, then individual nodes would have to react to each other in order to achieve a high fitness, and that this would result in reactive

and scalable behaviors.

For these two approaches, we initialized each simulation with a MANET comprising only 2 nodes. Then, in the case of the $SV2$ treatment, we unconditionally added 2 nodes to the simulation environment every 10s, and ran the simulation for a total of 120s (48 total nodes), at which point fitness was calculated. The second treatment, $SV2G$, was configured identically, except that nodes were added only if the existing network had a normalized fitness greater than 80%. This second treatment was inspired by the idea of "building blocks" in genetic algorithms [50], where instead of a building block comprising a series of co-located bits, here the building block is a behavior for a smaller network. In both cases, we used the fitness function shown in Equation 6.2. We note that for the $SV2G$ treatment, normalized fitness was always calculated based on the maximum possible number of nodes that could have been controlled (i.e., 48), instead of the number that were controlled based on the number of nodes added after 80% fitness had been reached.

Figure 6.19 plots the mean normalized fitness of the dominant individual per generation over 30 trials of each of these two treatments. In general, fitnesses and evolved behaviors under these two treatments were poor. In the $SV2$ treatment, where nodes were unconditionally added, a common behavior was for nodes to slowly move across the grid, relying on the addition of new (also slow-moving) nodes to achieve higher fitness.

The poor performance of the $SV2G$ treatment, where nodes were added only after a fitness of 80% was reached by the existing network, is explained by the small number of nodes that were being controlled: In the best case overall, only 12 nodes were controlled following 200 generations of evolution. However, we did notice that the more successful behaviors appeared to depend on communication among individuals, which we tested by

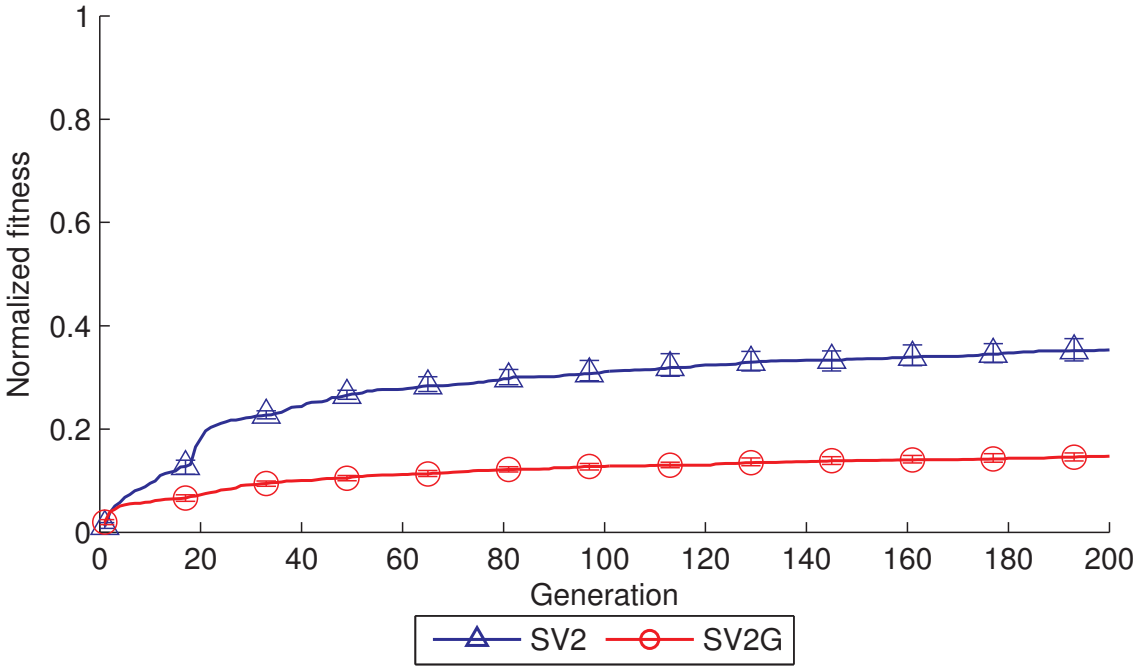


Figure 6.19: Normalized fitness for treatments where nodes are added to the simulation unconditionally ($SV2$), and conditionally based on the existing network reaching 80% of its possible fitness ($SV2G$).

turning communication off and reevaluating the dominant evolved solutions (this was not the case in the $SV2$ treatment). Figure 6.20 depicts one such reactive behavior, showing the position and velocity of each node during 20s of simulation time. Here, 2 nodes start along the $x-$axis, and quickly move to and settle in different nearby cells, achieving greater than 80% fitness and triggering the addition of more nodes. Then, as additional nodes join the network, they adapt to each others' presence, and spread out along the $x-$ and $y-$axes to occupy different cells.

This property, where individuals react to each other locally without regard to a global pattern, is an example of *self-organization*. There are numerous definitions for self-organization. For example, Haken [163] states that, "A system is self-organizing if it acquires a spatial, temporal, or functional structure without specific interference from the outside." Another

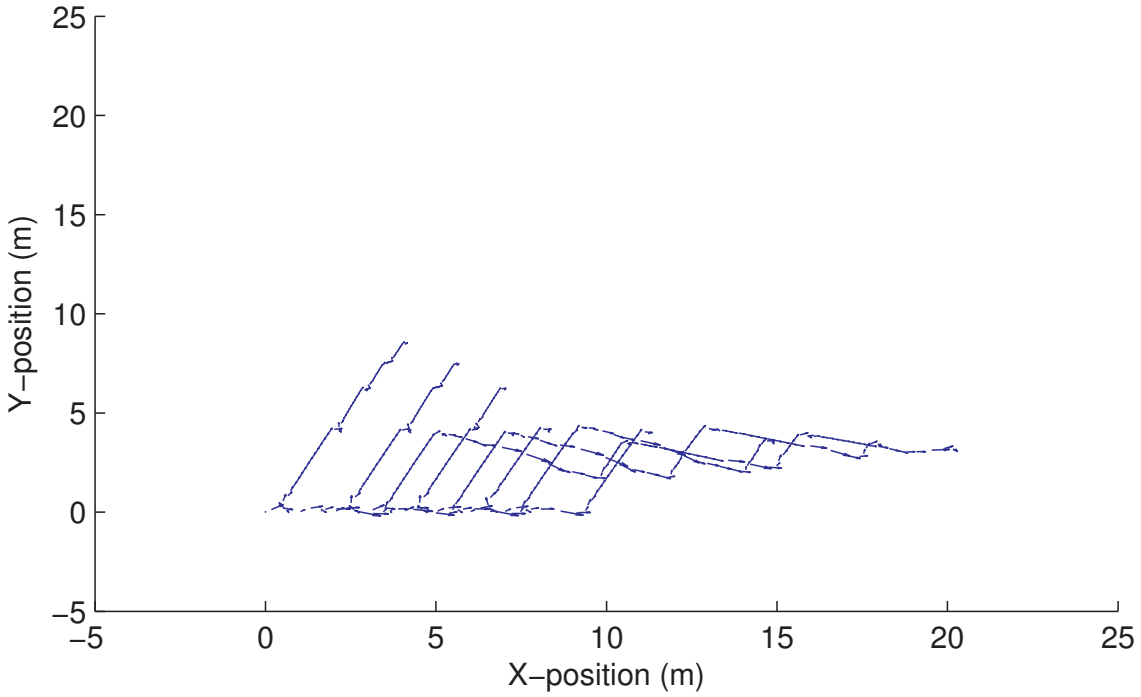


Figure 6.20: Velocity and positions of an example individual evolved under the $SV2G$ treatment.

definition is provided by Camazine [15]: "Self-organization is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system." Likewise, there have been numerous proposals for how to measure self-organization [164, 165, 166, 167, 168, 169, 170]. However, in the context of evolving a solution to the grid-coverage problem, what is needed is a way to ensure that the neural networks produced via evolution are communicating to solve the overall problem. We thus define *operational self-organization*, $S_{op}(x)$, of a neural network x, as:

$$S_{op}(x) = \frac{f(x) - f_{nc}(x)}{f(x) + f_{nc}(x)} \tag{6.4}$$

where $f(x)$ is the fitness of the neural network x with communication among agents enabled and $f_{nc}(x)$ is fitness with communication among agents disabled. $S_{op}(x)$ has the interesting

property that values greater than zero indicate the presence of self-organization (communication is beneficial), while values less than or equal to zero indicate the lack of self-organization (communication is harmful or neutral).

We then calculated operational self-organization for the dominant solutions from all of the treatments previously presented, which are shown in Figure 6.21. Surprisingly, here we see that the original fast-moving, highly entropic treatment ($FF16$) also had the highest degree of operational self-organization. Also surprising is that the $SF16$ treatment, which rewarded for reduced velocity and had only 3 bits of entropy, exhibited the least self-organization. This result, where stable solutions had the least self-organization, led us to include operational self-organization as a component of fitness, as will be seen shortly. First, however, let us consider scalability.

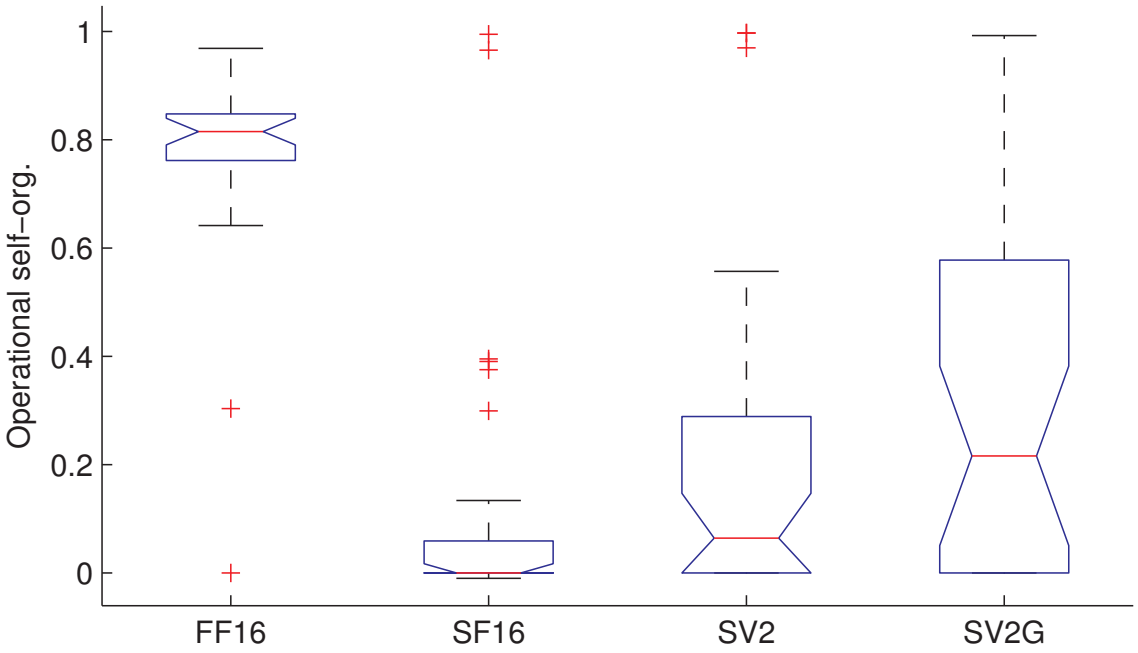


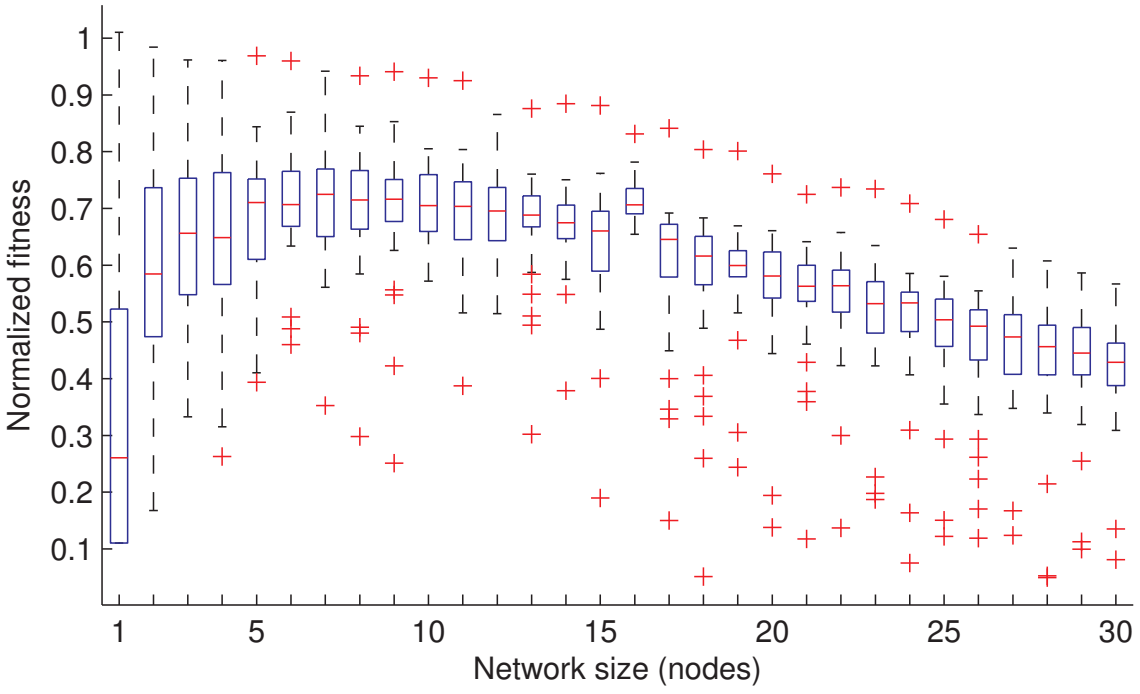
Figure 6.21: Operational self-organization of various treatments. Notably, the treatment with the greatest entropy ($FF16$) also exhibited the most self-organizatin.

6.3.4 Scalable Networks

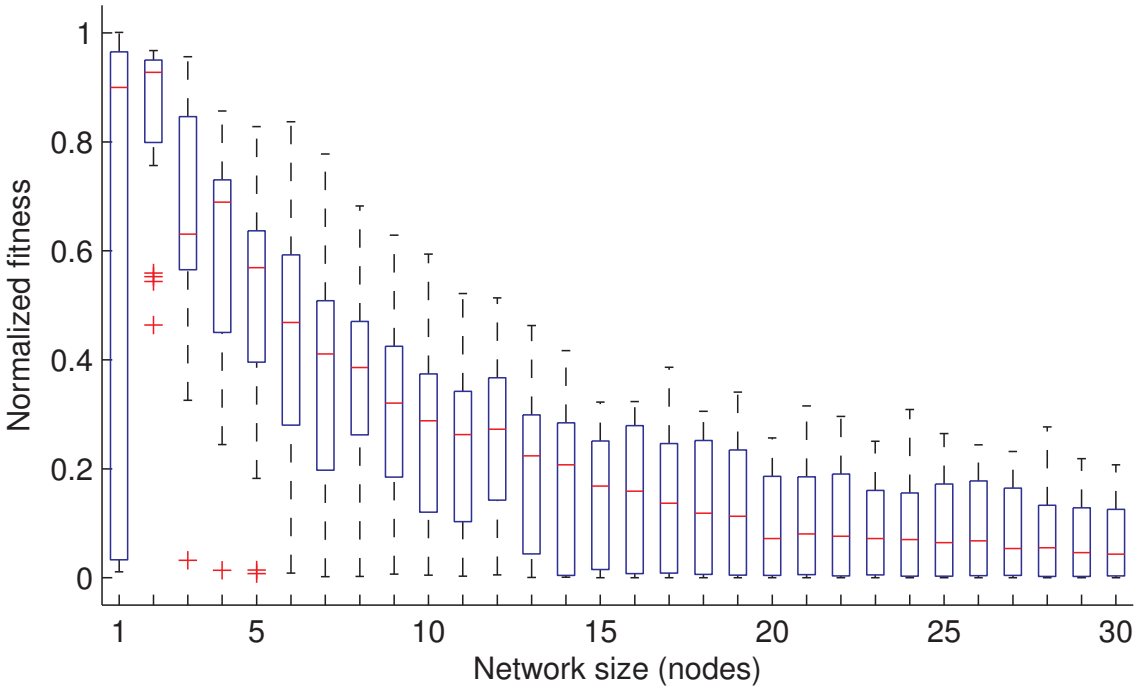
Scalability remains a key challenge not only for multi-agent systems [153], but also for many types of distributed systems [16]. Ideally, we wish to discover controllers that are scalable to large numbers of nodes and that can gracefully incorporate new nodes and withstand the removal of existing nodes. While the tendency for a given neural network controller to exhibit scalable behaviors can be difficult to capture analytically, we can measure it operationally, in a manner similar to the measure for operational self-organization presented above.

Figure 6.22 depicts normalized fitness for the dominant individuals from two different treatments on fixed-size networks of 1 to 30 nodes. These box plots show how fitness changes based on the number of nodes that are being controlled. In Figure 6.22(a), the $FF16$ treatment, we note a slight discontinuity at 16 nodes; this corresponds to the only network size that was used during evolution. Figure 6.22(b), the $SV2G$ treatment, shows very high (near-optimal) fitness for small networks, but performance quickly falls off as the number of nodes is increased. In both cases, we see that performance declines as additional nodes are available for control, though $FF16$ declines in performance more slowly than $SV2G$. We observe that the median performance of the 16-node case in the $FF16$ treatment is greatest, while the $SV2G$ treatment performs best overall when controlling small networks of 6 or fewer nodes. These results are consistent with the conditions that they experienced during evolution.

Based on the plots shown in Figure 6.22, we devised a heuristic for measuring scalability of neural networks that control the nodes of a MANET. Specifically, we define the approximate



(a)



(b)

Figure 6.22: Normalized fitness vs. network size for the dominant individuals from the $FF16$ treatment (a) and $SV2G$ treatment (b).

scalability, $C(x)$, of a neural network x as:

$$C(x) = \frac{\text{trapz}_{i=1}^{|S|} f(x, s_i)}{|S|} \tag{6.5}$$

where x is the neural network being evaluated, $S = \{s_1, \ldots, s_n\}$ is a set of network sizes over which x is evaluated, $f(x, s_i)$ is the normalized fitness achieved by a network of size s_i where each node is controlled by neural network x, and trapz$(\cdot)$ calculates the trapezoidal area (trapz$(\cdot)$ is an approximation of the definite integral). In essence, $C(x)$ is the area under the curves shown in Figure 6.22, scaled to the range [0..1]. We note that this definition of scalability is dependent on the set of network sizes (S) being evaluated, but for sufficient S, $C(x)$ will approach 1 for scalable neural network controllers.

Figure 6.23 depicts the scalability of the final dominants of the four treatments previously described. As with self-organization, here we again see that the $FF16$ treatment produced the most scalable solutions, though the relative ordering of the remaining treatments is different here than it was for self-organization.

We next performed an analysis of correlation of the different values of entropy, self-organization, and scalability that were measured from the dominant individuals from each of these four treatments. First, we found no statistically significant correlation between entropy and self-organization (Spearman rank correlation, $\rho = -0.007, p = 0.968$), which indicated that they are independent characteristics, and can thus be included as separate components of fitness functions. Second, we found a negative correlation between entropy and scalability ($\rho = -0.454, p = 0.011$), which reinforced the idea that a reduction in entropy is a desirable property in our search for scalable behaviors (as entropy decreased, scalability increased). Finally, we found a positive correlation between self-organization and scalability

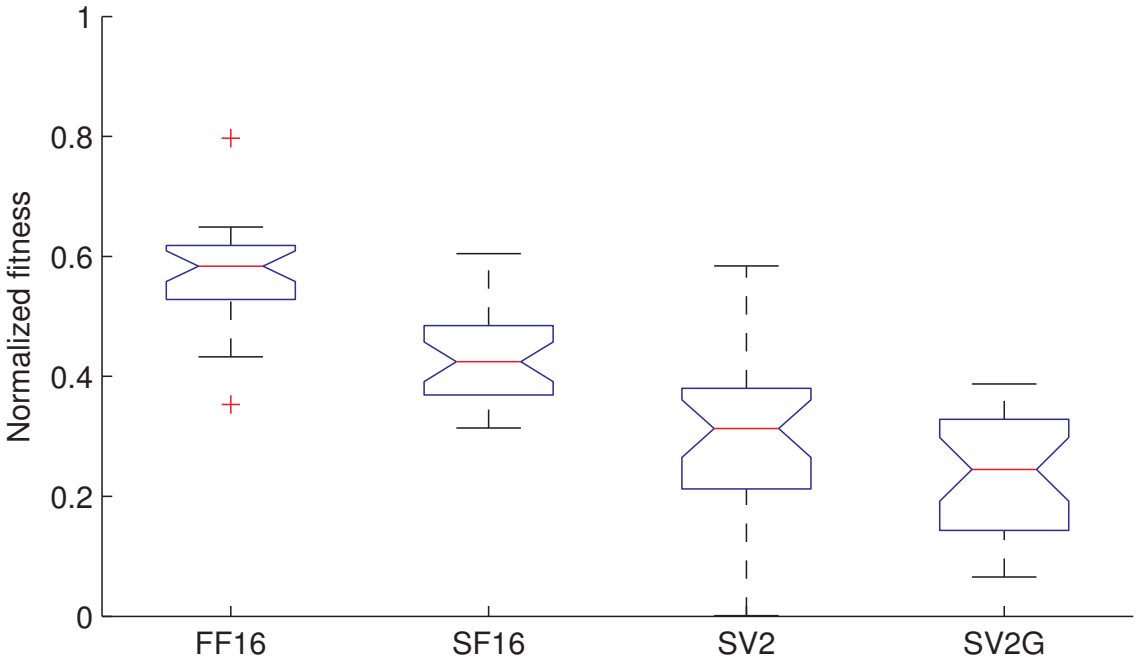


Figure 6.23: Scalability of four treatments, where treatment $FF16$, which also exhibited the most entropy and self-organization, is more scalable than other approaches.

($\rho = 0.376, p = 0.040$), which indicates that self-organization may be an important "building block" for scalable behaviors. Therefore, in the next experiment, we present an approach that reduces entropy while simultaneously increasing self-organization and scalability.

6.3.5 Stable, self-organizing, and scalable networks

In this section we present our final experiment on using neuroevolution to evolve MANET controllers, and introduce one method for evolving controllers that produce stable, self-organizing, and scalable behaviors. Similar to previous treatments, the fitness function presented here includes a component for reducing average speed (which has the side-effect of reducing entropy), as well as components for increasing self-organization and scalability. Specifically, we define the fitness function, $ESS(x)$, where x is the neural network being

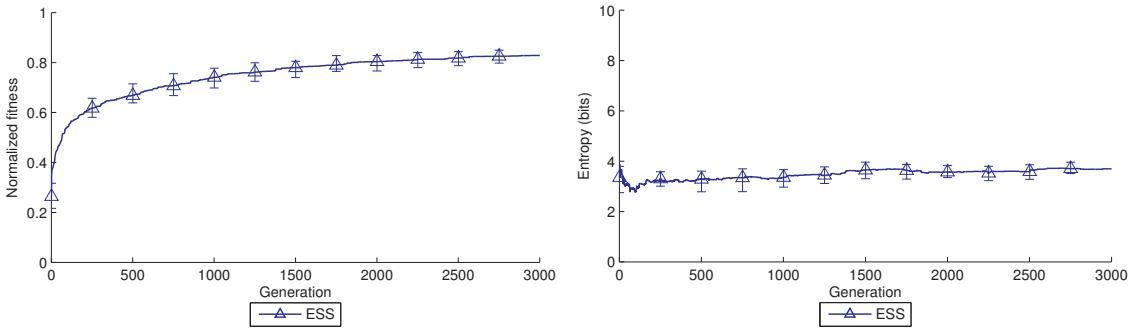
evaluated, as:

$$ESS(x) = \begin{cases} SF(x) + \max(0, S_{op}(x) & \text{if } S_{op}(x) \leq 0 \\ SF(x) + \max(0, S_{op}(x) + C(x) & \text{if } S_{op}(M) > 0 \end{cases} \tag{6.6}$$

where $SF(x)$ is performance on the grid-coverage problem, as defined earlier in Equation 6.1; $S_{op}(x)$ is self-organization, as defined in Equation 6.4; and $C(M)$ is the measured scalability of x tested on networks of size 8, 16, 24, and 32, as described in Equation 6.5.

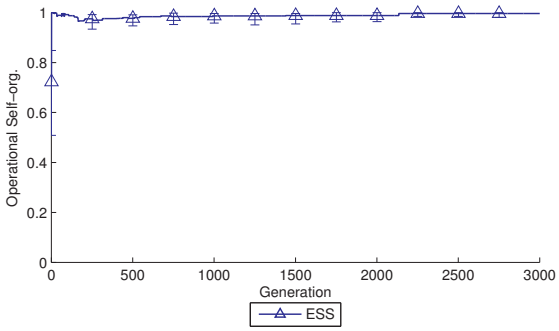
Evaluating the fitness of a neural network x using Equation 6.6 proceeds as follows. First, performance of a network comprising 8-nodes controlled by x is evaluated on the grid-coverage problem using Equation 6.1. Then, performance is re-evaluated in a simulation in which communication is disabled, allowing us to calculate operational self-organization using Equation 6.4. If the resulting value for S_{op} is positive, we then evaluate x on networks of size 16, 24, and 32 (communication is enabled in these simulations). The normalized fitness values for all 4 different network sizes are then used in Equation 6.5 to calculate the approximate scalability of x.

Figure 6.24 depicts mean normalized fitness, entropy, operational self-organization, and scalability per generation for the dominant individuals from 30 trials of the ESS treatment. This treatment was allotted 3,000 generations primarily because fitness was still improving after 200 generations (this fitness function is also more complex than those previously presented, thus we would expect that more generations be required). In these plots, we consistently low entropy (Figure 6.24(b), consistently high operational self-organization (Figure 6.24(c)), and scalability that slowly improves (Figure 6.24(d)). We note that the slow improvement in fitness is primarily driven by scalability.

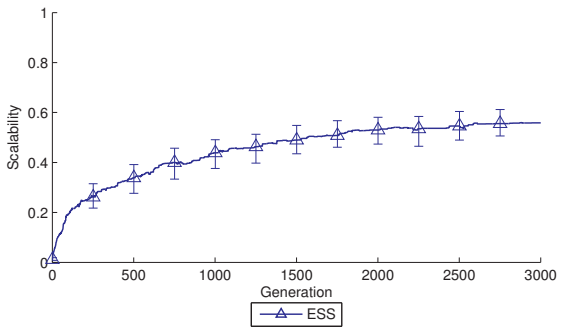


(a) Normalized fitness.

(b) Entropy.



(c) Operational self-organization.



(d) Scalability.

Figure 6.24: Characteristics of the dominant individuals evolved under treatment ESS.

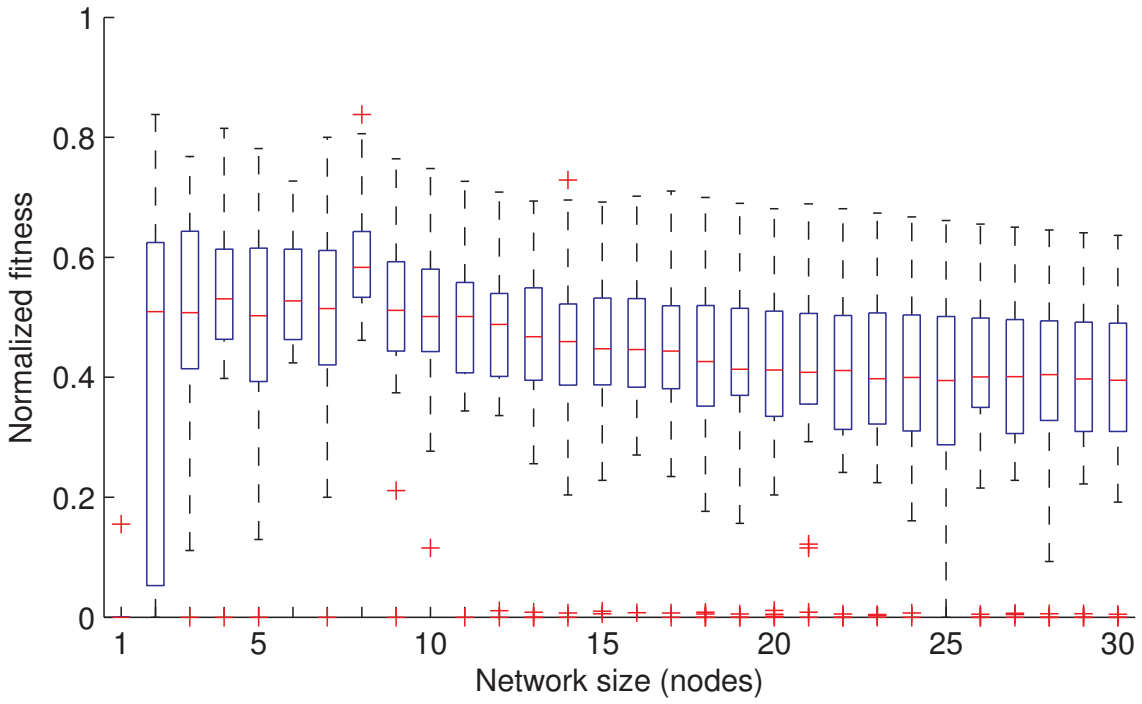
Interestingly, we did observe that rewarding explicitly for scalability had an immediate effect, even after only 200 generations. In Figure 6.25, we see normalized fitness vs. network size after 200 and after 3,000 generations. Compared to Figure 6.22, here we see much more consistent performance across a variety of network sizes, and a much less pronounced decline in fitness as network sizes increase.

Figure 6.26 depicts the position and velocity of a 16-node network during 20s of simulation for one of the dominant individuals evolved here. Figure 6.26(a) depicts movement when nodes are able to communicate. Here, nodes begin arrayed along the $x-$axis, and their first movements are away from the grid. When they are able to communicate, a subset rapidly advance across the grid, reversing direction just short of its boundary. The remaining nodes also advance, somewhat more slowly, stopping near the middle of the grid. Figure 6.26(b)

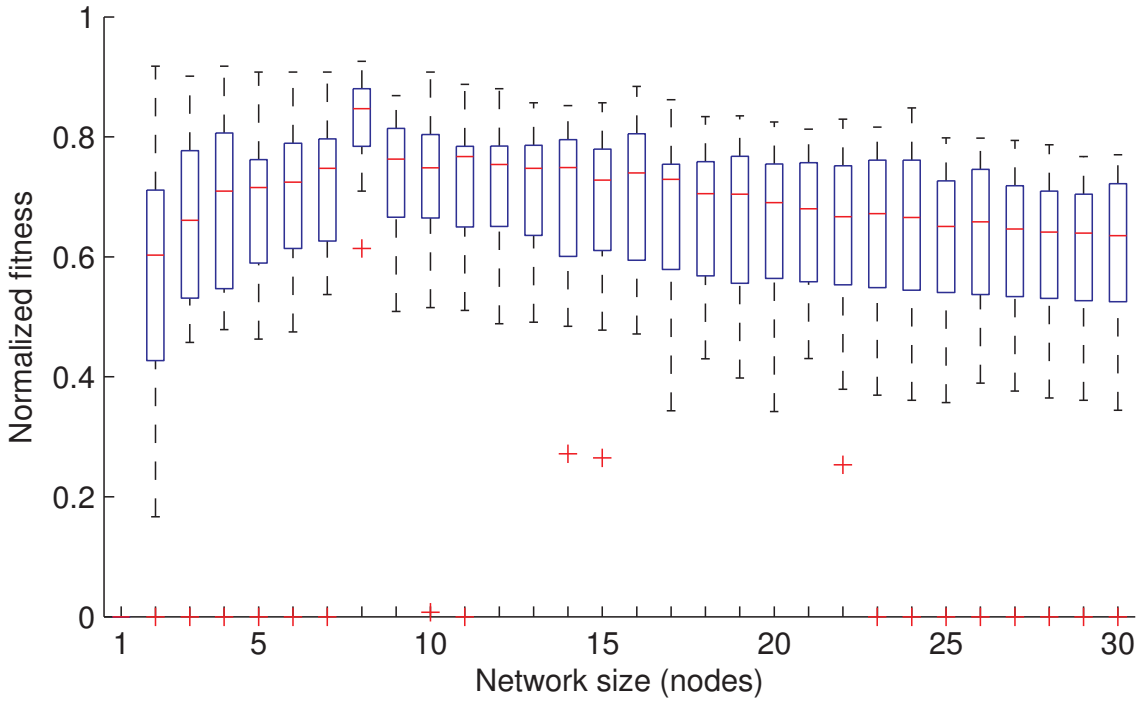
depicts movement when nodes are unable to communicate, as during the later half of the calculation of operational self-organization. As before, nodes begin by slowly moving away from the grid, but in the absence of communication this behavior does not change. Nodes are thus actively using the presence of neighbors to alter their behavior.

6.4 Conclusion

In this chapter, we have explored the evolution of a compound distributed behavior, specifically the evolution of controllers for mobile ad hoc networks. We specifically focused on a coverage-related problem that required coordination among network nodes. In our first study, we evaluated multiple different neuroevolutionary systems, and found that the key factor in network performance was the number of nodes under control. Building on this result, our next study focused on the question of scalability, and we found that concrete measures of stability, self-organization, and scalability, when integrated into evolutionary algorithms, hold promise for the design multi-agent systems.

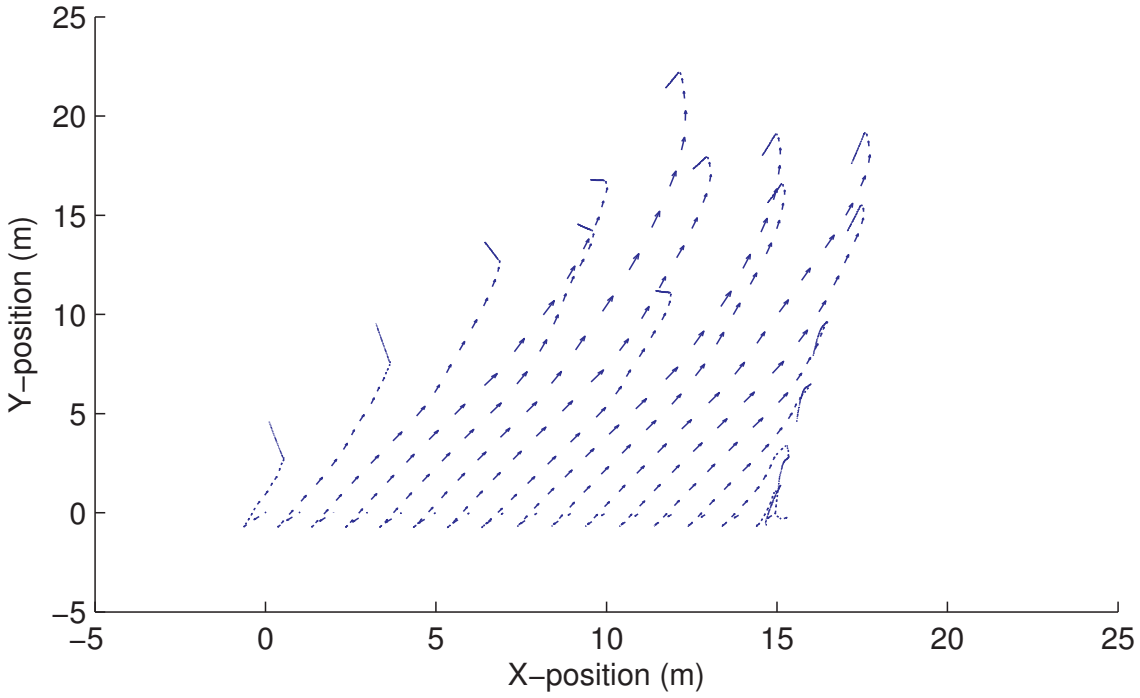


(a)

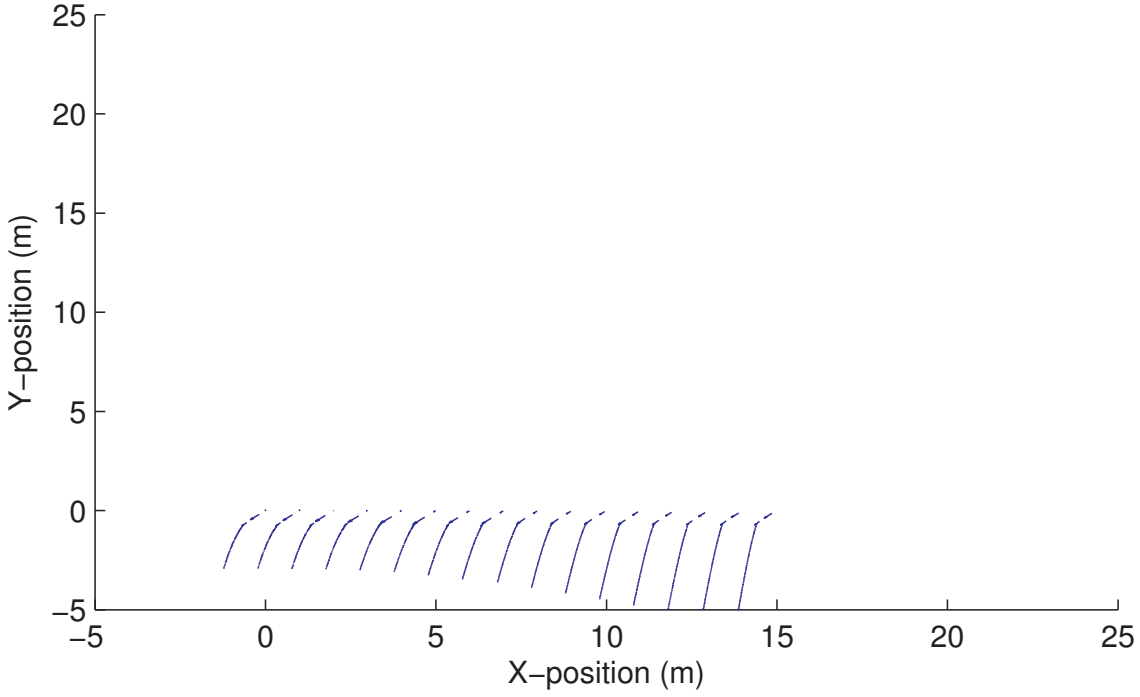


(b)

Figure 6.25: Normalized fitness vs. network size for the dominant individuals from after 200 (a) and 3,000 (b) generations under the *ESS* treatment.



(a)



(b)

Figure 6.26: Velocity and positions of an example individual evolved under the *ESS* treatment with communication enabled (a), and with communication disabled (b).

Chapter 7

Conclusion and Future Work

This dissertation described a study in the evolution of distributed behavior, where we used evolutionary algorithms to discover a variety of behaviors for distributed computing systems. Our results have demonstrated that evolutionary algorithms can discover candidate solutions (i.e., digital organisms and artificial neural networks) that exhibit each of three different classes of distributed behavior: data-driven behaviors, where semantically meaningful data is transmitted between individuals; temporal behaviors, which are based on the relative timing of individuals' actions; and structural behaviors, which are responsible for maintaining the underlying communication network connecting individuals.

In our study of data-driven behavior, we began with an investigation of leader election, where we first rewarded individual organisms based on a set of preconceived ideas about how leader election should be performed [171]. We then progressed to studies that used multilevel selection, and relaxed the individual-level selection pressures in favor of group and multilevel selection [172]. We concluded with a study of a general form of distributed consensus, where digital evolution in fact discovered a novel heuristic for probabilistic consensus [173].

We next explored temporal behaviors by way of the dual problems of synchronization and desynchronization. In this study, we demonstrated that digital evolution, when provided with primitives based on biological fireflies and local information, was able to discover an adaptive frequency strategy for synchronization, and that a simple change to a group-level selection pressure was able to evolve the opposite behavior, desynchronization [174, 175].

Structural behaviors are those that concern the underlying communication network. In our study of structural behavior, we explored the relationship between evolution of network construction and data distribution behaviors. We found that digital organisms were able to construct networks with various characteristics [94], and that they could perform data distribution on these networks even in the presence of node and message loss [176]. This result suggests that digital evolution could be used for a new class of distributed algorithm that automatically builds networks specifically tuned for their application.

In our final study, we addressed the evolution of compound behavior, specifically the control of a mobile ad hoc network [54]. This problem required the simultaneous evolution of all three classes of distributed behavior already mentioned, and we found that including measures of stability, self-organization, and scalability were critical factors for the evolution of effective controllers for these mobile networks.

Many of the studies described in this dissertation were motivated by observations of natural systems. For example, there is a clear relationship between our study of temporal behavior, specifically synchronization, and the synchronization of firefly flashes in nature. Likewise, our studies of data-driven and structural behaviors were similarly motivated by observations related to communication among, and the edifices constructed by, groups of organisms.

While evolutionary algorithms have long been used for engineering optimization, it is hoped that the results of this dissertation will inspire the next-generation of distributed computing systems, where engineers will not only leverage observations of nature today, but also harness the power of evolution by natural selection to help us build the distributed systems of tomorrow.

Future work. The research presented in this dissertation has shown that digital evolution, specifically Avida, can produce a variety of distributed behaviors. These studies also suggest the following future research:

Network simulation. Much of our research was conducted using Avida, which we adapted to evolve distributed behaviors. The approach taken to evolve consensus, (de-)synchronization, and network construction could all be replicated using a traditional evolutionary algorithm (for example, a genetic program) and network simulator (such as ns2). While this is likely to entail substantial effort, such research could produce distributed algorithms that are more easily adapted to real-world systems.

Protocol evolution. Many of the evolved behaviors for network construction and data distribution are stateful, which is a common feature of network protocols. This suggests that EAs could in fact be used to discover network protocols directly.

Individual-level selection studies. Our early work on leader election used individual-level selection pressures to evolve a very specific form of leader election. Our later worked abandoned this approach in favor of group-based selection pressures, which we found to be more effective at evolving distributed behavior. However, it can be argued that selection in nature only operates on individuals. It is possible that we could use individual-level selection pressures, based on group performance, to evolve distributed behaviors.

Evolution of multicellularity. The extensions to Avida for this research include many features that bridge the gap between the evolution of an individual and the evolution of a group, e.g., our study of leader election via multilevel selection. With minor extension, it is thus possible that Avida could be used to study the transition of individual organisms to multicellular digital life. While a substantial undertaking, such research has the potential to help us unravel questions related to transitions in biological evolution [177].

Multi-agent systems. Our study of the evolution of controllers for MANETs identified three characteristics which were found beneficial (stability, self-organization, and scalability). Further study of these are needed, in different evolutionary algorithms, more complex environments, and of course, in real mobile ad hoc networks.

BIBLIOGRAPHY

Bibliography

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM* (31, ed.), vol. 4, pp. 149–160, 2001.

[2] M. Ripeanu and I. Foster, "Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems," in *International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 85–93, 2002.

[3] J. Lohn, G. Hornby, and D. Linden, "An evolved antenna for deployment on NASA's space technology 5 mission," in *Genetic Programming Theory and Practice II* (U.-M. O'Reilly, T. Yu, R. Riolo, and B. Worzel, eds.), pp. 301–313, Springer, 2004.

[4] J. Lohn, G. S. Hornby, and D. Linden, "Evolution, re-evolution, and prototype of an x-band antenna for nasa's space technology 5 mission," in *International Conference on Evolvable Systems: From Biology to Hardware*, 2005.

[5] M. E. Davey and G. A. O'Toole, "Microbial biofilms: from ecology to molecular genetics," *Microbiology and Molecular Biology Reviews*, vol. 64, no. 4, pp. 847–867, 2000.

[6] D. Grimaldi and M. S. Engel, *Evolution of the Insects*. Cambridge University Press, 2005.

[7] B. Holldobler and E. O. Wilson, *The Ants*. Harvard University Press, 1990.

[8] 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.

[9] C. List, "Democracy in animal groups: a political science perspective," *Trends in Ecology and Evolution*, vol. 19, no. 4, pp. 168–169, 2004.

[10] 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, 2005.

[11] E. Fernandez-Juricic, S. Siller, and A. Kacelnik, "Flock density, social foraging, and scanning: an experiment with starlings," *Behavioral Ecology*, vol. 15, no. 3, pp. 371–379, 2004.

[12] D. H. Owings and E. S.Morton, *Animal Vocal Communication: A New Approach*. Cambridge University Press, 1998.

[13] K. von Frisch, *The Dance Language and Orientation of Bees*. Harvard University Press, 1967.

[14] F. C. Dyer, "The biology of the dance language," *Annual Review of Entomology*, vol. 47, no. 1, pp. 917–949, 2002.

[15] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraula, and E. Bonabeau, *Self-organization in biological systems*. Princeton University Press, 2003.

[16] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, *Ultra-Large-Scale Systems - The Software Challenge of the Future*. Software Engineering Institute, Carnegie Mellon, June 2006.

[17] W. Wolf, "Cyber-physical systems," *IEEE Computer*, vol. 42, no. 3, 2009.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," in *IEEE Computer*, vol. 36, pp. 41–50, 2003.

[19] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen, eds., *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, Lecture Notes in Computer Science, Springer, 2005.

[20] 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.

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

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

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

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

[25] C. Emmeche, "Does a robot have an Umwelt? Reflections on the qualitative biosemiotics of Jakob von Uexküll," *Semiotica*, no. 134, pp. 653–693, 2001.

[26] A. Vicini and D. Quagliarella, "Airfoil and wing design through hybrid optimization strategies," *AIAA journal*, vol. 37, no. 5, pp. 634–641, 1999.

[27] Y. Xie and G. Steven, "Optimal design of multiple load case structures using an evolutionary procedure," *Engineering Computations*, vol. 11, no. 4, pp. 295 – 302, 1994.

[28] F. Ferrandi, P. Lanzi, G. Palermo, C. Pilato, D. Sciuto, and A. Tumeo, "An evolutionary approach to area-time optimization of FPGA designs," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (ICSAMOS)*, pp. 145–152, 2007.

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

[30] J. Clune, C. Ofria, and R. T. Pennock, "Investigating the emergence of phenotypic plasticity in evolving digital organisms," in *European Conference on Artificial Life (ECAL)*, pp. 74–83, 2007.

[31] H. J. Goldsby, D. B. Knoester, J. Clune, P. K. McKinley, and C. Ofria, "The evolution of division of labor," in *Proceedings of the European Conference on Artificial Life (ECAL)*, 2009.

[32] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," in *Wireless Sensor Networks*, vol. 2920 of *Lecture Notes in Computer Science*, pp. 307–322, Springer Berlin / Heidelberg, 2004.

[33] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, "Firefly-inspired heartbeat synchronization in overlay networks," in *Proceedings of Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 77–86, 2007.

[34] D. Comer, *Internetworking with TCP/IP: principles, protocols, and architecture*. Prentice Hall, 2006.

[35] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 131–145, 2001.

[36] J. Buck, "Synchronous Rhythmic Flashing of Fireflies, II," *The Quarterly Review of Biology*, vol. 63, no. 3, pp. 265–289, 1988.

[37] G. Weiss, ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[38] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications*. Springer, 2007.

[39] F. Mondada, G. C. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo, "SWARM-BOT: A swarm of autonomous mobile robots with self-assembling capabilities," in *Proceedings of the Workshop on Self-organisation and Evolution of Social Behaviour*, 2002.

[40] A. Tero, S. Takagi, T. Saigusa, K. Ito, D. P. Bebber, M. D. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki, "Rules for Biologically Inspired Adaptive Network Design," *Science*, vol. 327, no. 5964, pp. 439–442, 2010.

[41] R. J. Anthony, "An autonomic election algorithm based on emergence in natural systems," *Integrated Computer-Aided Engineering*, vol. 13, no. 1, 2006.

[42] M. Mnif and C. Müller-Schloer, "Quantitative emergence," in *Proceedings of the IEEE Mountain Workshop on Adaptive and Learning Systems*, 2006.

[43] R. P. Würtz, ed., *Organic Computing*. Understanding Complex Systems, Springer, 2008.

[44] P. Snyder, R. Greenstadt, and G. Valetto, "Myconet: A Fungi-inspired Model for Superpeer-based Peer-to-Peer Overlay Topologies," in *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2009.

[45] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," in *Proceedings of the Workshop on Engineering Self-Organising Applications (ESOA)*, pp. 1–15, 2005.

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

[47] B. Ermentrout, "An adaptive model for synchrony in the firefly *pteroptyx malaccae*," *Journal of Mathematical Biology*, vol. 29, no. 6, pp. 571–585, 1991.

[48] R. E. Mirollo and S. H. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM Journal on Applied Mathematics*, vol. 50, no. 6, pp. 1645–1662, 1990.

[49] A. Patel, J. Degesys, and R. Nagpal, "Desynchronization: The theory of self-organizing algorithms for round-robin scheduling," in *Proceedings of Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 87–96, 2007.

[50] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2003.

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

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

[53] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based indirect encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, 2009.

[54] D. B. Knoester, H. J. Goldsby, and P. K. McKinley, "Neuroevolution of mobile ad hoc networks," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 603–610, 2010.

[55] M. Mitchell and C. E. Taylor, "Evolutionary computation: An overview," *Annual Review of Ecology and Systematics*, vol. 30, no. 1, pp. 593–616, 1999.

[56] J. M. Smith, "Byte-sized evolution," *Nature*, vol. 355, pp. 772–773, 1992.

[57] R. E. Lenski, "Evolution in action: a 50,000-generation salute to charles darwin," *Microbe*, vol. 6, pp. 30–33, 2011.

[58] R. M. Axelrod and W. D. Hamilton, "The evolution of cooperation," *Science*, vol. 211, no. 4489, pp. 1390–1396, 1981.

[59] D. S. Wilson, "A theory of group selection," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 72, pp. 143–146, January 1975.

[60] A. Traulsen and M. A. Nowak, "Evolution of cooperation by multilevel selection," *Proceedings of the National Academy of Sciences*, vol. 103, no. 29, pp. 10952–10955, 2006.

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

[62] J. J. Bull and W. R. Rice, "Distinguishing mechanisms for the evolution of cooperation," *Journal of Theoretical Biology*, vol. 149, no. 1, pp. 63–74, 1991.

[63] R. Dawkins, *The extended phenotype*. Oxford University Press, 1982.

[64] R. Dawkins, *The selfish gene*. Oxford University Press, 2006.

[65] R. M. Axelrod and D. Dion, "The further evolution of cooperation," *Science*, vol. 242, no. 4884, pp. 1385–1390, 1988.

[66] D. Floreano, S. Mitri, S. Magnenat, and L. Keller, "Evolutionary conditions for the emergence of communication in robots," *Current Biology*, vol. 17, no. 6, pp. 514–519, 2007.

[67] 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.

[68] D. Helbing and W. Yu, "The outbreak of cooperation among success-driven individuals under noisy conditions," *Proceedings of the National Academy of Sciences*, vol. 106, no. 10, pp. 3680–3685, 2009.

[69] M. A. Nowak, "Five rules for the evolution of cooperation," *Science*, vol. 314, no. 5805, pp. 1560–1563, 2006.

[70] G. M. Werner and M. G. Dyer, "Evolution of communication in artificial organisms," in *Artificial Life II*, pp. 659–687, 1992.

[71] G. M. P. O'Hare and N. R. Jennings, eds., *Foundations of Distributed Artificial Intelligence*. Wiley-Interscience, 1996.

[72] C. Baray, "Evolving cooperation via communication in homogeneous multi-agent systems," in *Proceedings of the International Conference on Intelligent Information Systems*, 1997.

[73] K. Wagner, "Cooperative strategies and the evolution of communication," *Artificial Life*, vol. 6, no. 2, pp. 149–179, 2000.

[74] K. Wagner and J. A. Reggia, "Evolving consensus among a population of communicators," *Complexity International*, vol. 9, 2002.

[75] 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.

[76] J. Clune, H. J. Goldsby, C. Ofria, and R. T. Pennock, "Selective pressures for accurate altruism targeting: evidence from digital evolution for difficult-to-test aspects of inclusive fitness theory," *Proceedings of the Royal Society B: Biological Sciences*, 2010. (Accepted for publication, early access version available.).

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

[78] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, 2002.

[79] C. Ofria and C. Adami, "Evolution of genetic organization in digital organisms," in *Proceedings of DIMACS Workshop on Evolution as Computation*, p. 296, 1999.

[80] M. Guimaraes and L. Rodrigues, "A genetic algorithm for multicast mapping in publish-subscribe systems," in *Second IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 67–74, 2003.

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

[82] L. Yamamoto and C. F. Tschudin, "Experiments on the automatic evolution of protocols using genetic programming," in *Proceedings of the IFIP Workshop on Autonomic Communication (WAC)*, 2005.

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

[84] G. Baldassarre, D. Parisi, and S. Nolfi, "Distributed coordination of simulated robots based on self-organization," *Artificial Life*, vol. 12, no. 3, 2006.

[85] D. Marocco and S. Nolfi, "Self-organization of communication in evolving robots," in *Proceedings of the Conference on Artificial Life (ALIFE)*, pp. 178–184, 2006.

[86] J. C. Bongard, "The impact of jointly evolving robot morphology and control on adaptation rate," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

[87] M. A. Potter, L. A. Meeden, and A. C. Schultz, "Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists," in *Proceedings of the International Conference on Artificial Intelligence*, 2001.

[88] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler, "Co-evolving soccer softbot team coordination with genetic programming," in *Robot Soccer World Cup I (RoboCup)*, 1997.

[89] W. Lee, J. Hallam, and H. H. Lund, "A hybrid GP/GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1996.

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

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

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

[93] S. Goings, J. Clune, C. Ofria, and R. T. Pennock, "Kin selection: The rise and fall of kin-cheaters," in *Proceedings of the Conference on Artificial Life (ALIFE)*, pp. 124–130, 2004.

[94] 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)*, pp. 217–224, 2008.

[95] 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)*, pp. 181–190, 2008.

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

[97] D. S. Wilson, "Introduction: Multilevel selection theory comes of age," *The American Naturalist*, vol. 150, pp. S1–S4, July 1997.

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

[99] M. J. W. Eberhard, "The evolution of social behavior by kin selection," *The Quarterly Review of Biology*, vol. 50, pp. 1–33, March 1975.

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

[101] T. D. Seeley, "Honey bee colonies are group-level adaptive units," *The American Naturalist*, vol. 150, pp. S22–S41, 1997.

[102] D. C. Queller and J. E. Strassmann, "Kin selection and social insects," *BioScience*, vol. 48, pp. 165–175, March 1998.

[103] M. Waibel, L. Keller, and D. Floreano, "Genetic Team Composition and Level of Selection in the Evolution of Cooperation," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 648–660, 2009.

[104] E. Eddy, "The germ line and development," *Developmental Genetics*, vol. 19, pp. 287–289, 1996.

[105] J. Hoogland, "Nepotism and alarm calling in the black-tailed prairie dog (Cynomys ludovicianus).," *Animal Behaviour*, vol. 31, no. 2, pp. 472–479, 1983.

[106] 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.

[107] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, Jan 2004.

[108] C. F. Camerer, *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press, 2003.

[109] 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.

[110] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[111] 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)*, pp. 398–407, 2007.

[112] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pp. 335–350, 2006.

[113] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817 – 840, 2004.

[114] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *Proceedings of the IEEE International Workshop on Sensor Network Protocols and Applications*, pp. 139–148, May 2003.

[115] E. Cohen, "Figuring immunity: Towards the genealogy of a metaphor," in *Singular Selves: Historical Issues and Contemporary Debates in Immunology* (A.-M. Moulin and A. Cambrosio, eds.), pp. 179–201, Elsevier, Amsterdam, 2001.

[116] T. D. Chandra, "Polylog randomized wait-free consensus," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 166–175, 1996.

[117] 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, pp. 215–233, 2007.

[118] W. Ren, R. W. Beard, and E. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control Systems Magazine*, Jan 2007.

[119] A. Campbell and A. S. Wu, "On the significance of synchroneity in emergent systems," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 449–456, 2009.

[120] E. M. Zechman and S. R. Ranjithan, "Multipopulation cooperative coevolutionary programming (MCCP) to enhance design innovation," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1641–1648, 2005.

[121] O. Giel and P. K. Lehre, "On the effect of populations in evolutionary multi-objective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 651–658, 2006.

[122] J. C. Bongard, "The Legion System: A novel approach to evolving hetrogeneity for collective problem solving," in *Proceedings of the European Conference on Genetic Programming*, pp. 16–28, 2000.

[123] C. Yong and R. Miikkulainen, "Cooperative coevolution of multi-agent systems," Tech. Rep. AI-01-287, University of Texas at Austin, Austin, TX, February 2001.

[124] R. Jain, M. Rivera, and J. Lake, "Horizontal gene transfer among genomes: the complexity hypothesis," *Proceedings of the National Academy of Sciences*, vol. 96, no. 7, p. 3801, 1999.

[125] H. Ochman, J. Lawrence, E. Groisman, *et al.*, "Lateral gene transfer and the nature of bacterial innovation," *Nature*, vol. 405, no. 6784, pp. 299–304, 2000.

[126] K. E. Nelson, R. A. Clayton, and S. R. Gill et al., "Evidence for lateral gene transfer between archaea and bacteria from genome sequence of thermotoga maritima," *Nature*, vol. 399, no. 6734, pp. 323–329, 1999.

[127] J. O. Andersson, "Lateral gene transfer in eukaryotes," *Cellular and Molecular Life Sciences*, vol. 62, pp. 1182–1197, 2005.

[128] M. Lessl and E. Lanka, "Common mechanisms in bacterial conjugation and ti-mediated t-dna transfer to plant cells," *Cell*, vol. 77, no. 3, pp. 321–324, 1994.

[129] M. Slatkin, "Gene flow in natural populations," *Annual Review of Ecology and Systematics*, vol. 16, no. 1, pp. 393–430, 1985.

[130] M. Slatkin, "Gene flow and the geographic structure of natural populations," *Science*, vol. 236, no. 4803, p. 787, 1987.

[131] G. R. Abecasis, S. S. Cherny, W. O. Cookson, and L. R. Cardon, "Merlin–rapid analysis of dense genetic maps using sparse gene flow trees," *Nature Genetics*, vol. 30, no. 1, pp. 97–101, 2002.

[132] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[133] C. Pittendrigh, "Circadian rhythms and the circadian organization of living systems," in *Cold Spring Harbor Symposia on Quantitative Biology*, vol. 25, pp. 159–184, 1960.

[134] E. E. Southwick and R. F. A. Moritz, "Social synchronization of circadian rhythms of metabolism in honeybees (apis mellifera)," *Physiological Entomology*, vol. 12, no. 2, pp. 209–212, 1987.

[135] P. Backwell, M. Jennions, N. Passmore, and J. Christy, "Synchronized courtship in fiddler crabs," *Nature*, vol. 391, no. 6662, pp. 31–32, 1998.

[136] J. M. Palva, S. Palva, and K. Kaila, "Phase synchrony among neuronal oscillations in the human cortex," *Journal of Neuroscience*, vol. 25, no. 15, 2005.

[137] O. Oullier, G. C. de Guzman, K. J. Jantzen, J. Lagarde, and J. A. S. Kelso, "Social coordination dynamics: Measuring human bonding," *Social Neuroscience*, vol. 3, no. 2, 2008.

[138] A. L. Christensen, R. O'Grady, and M. Dorigo, "From fireflies to fault-tolerant swarms of robots," *IEEE Transactions on Evolutionary Computation (TEC)*, vol. 13, pp. 754–766, Aug. 2009.

[139] V. Trianni and S. Nolfi, "Self-organizing sync in a robotic swarm: a dynamical system view," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, 2009.

[140] J. Korb, "Thermoregulation and ventilation of termite mounds," *Naturwissenschaften*, vol. 90, no. 5, pp. 212–219, 2003.

[141] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Computing Surveys*, vol. 37, no. 2, pp. 164–194, 2005.

[142] N. Wakamiya, K. Hyodo, and M. Murata, "Reaction-diffusion based topology self-organization for periodic data gathering in wireless sensor networks," in *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2008.

[143] S. Kondo and R. Asai, "A reaction-diffusion wave on the kin of the marine angelfish pomacanthus," *Nature*, vol. 376, pp. 765–768, 1995.

[144] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[145] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, October 1999.

[146] J. Fax, R. Murray, N. Syst, and C. W. Hills, "Information flow and cooperative control of vehicle formations," *IEEE Transactions on Automatic Control*, Jan 2004.

[147] R. Fierro, P. Song, A. Das, and V. Kumar, "Cooperative control of robot formations," *Applied Optimization*, vol. 66, pp. 73–93, 2002.

[148] P. Ogren, E. Fiorelli, and N. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Transactions on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, 2004.

[149] J. Cortes, S. Martinez, T. Karatas, F. Bullo, *et al.*, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[150] C. Cassandras and W. Li, "Sensor networks and cooperative control," *European Journal of Control*, vol. 11, no. 4-5, 2005.

[151] K. Y. Lim, "A performance analysis of an ad-hoc ocean sensor network," Master's thesis, Naval Postgraduate School, Monterey, California, USA, Dec 2006.

[152] W. L. Patterson, "Advanced refractive effects prediction system (areps)," in *IEEE Radar Conference*, 2007.

[153] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 648–660, 2005.

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

[155] G. Baldassarre, D. Parisi, and S. Nolfi, "Coordination and behavior integration in cooperating simulated robots," in *Proceedings of the Conference on Simulation of Adaptive Behavior*, pp. 385–394, 2004.

[156] S. Hauert, J.-C. Zufferey, and D. Floreano, "Evolved swarming without positioning information: an application in aerial communication relay," *Autonomous Robots*, vol. 26, no. 1, 2009.

[157] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley, "Evolving policy geometry for scalable multiagent learning," in *Proceedings of the Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[158] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2008.

[159] J. Clune, C. Ofria, and R. Pennock, "How a generative encoding fares as problem-regularity decreases," in *Proceedings of the Conference on Parallel Problem Solving From Nature (PPSN)*, 2008.

[160] J. Clune, B. Beckmann, R. Pennock, and C. Ofria, "Hybrid: A hybridization of indirect and direct encodings for evolutionary computation," in *Proceedings of the European Conference on Artificial Life (ECAL)*, 2009.

[161] J. Clune, C. Ofria, and R. Pennock, "The sensitivity of HyperNEAT to different geometric representations of a problem," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, 2009.

[162] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[163] H. Haken, *Information and Self-Organization: A Macroscopic Approach to Complex Systems*. Springer, Berlin, Heidelberg, 2006.

[164] D. Polani, "Measuring self-organization via observers," in *Advances in Artificial Life*, vol. 2801 of *Lecture Notes in Computer Science*, pp. 667–675, Springer Berlin / Heidelberg, 2003.

[165] C. R. Shalizi, K. L. Shalizi, and R. Haslinger, "Quantifying self-organization with optimal predictors," *Phys. Rev. Lett.*, vol. 93, no. 118701, 2004.

[166] E. Anceaume, X. Défago, M. Gradinariu, and M. Roy, "Towards a theory of self-organization," in *Proceedings of the Conference on Principles of Distributed Systems (OPODIS)*, pp. 191–205, 2005.

[167] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, "Adaptivity and self-organization in organic computing systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 5, September 2010.

[168] E. Cakar, M. Mnif, C. Muller-Schloer, U. Richter, and H. Schmeck, "Towards a quantitative notion of self-organisation," in *IEEE Congress on Evolutionary Computation (CEC)*, 2007.

[169] D. Yamins, "Towards a theory of "local to global" in distributed multi-agent systems (I and II)," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 183–190, 2005.

[170] M. Prokopenko, "Design vs. self-organization," in *Advances in Applied Self-organizing Systems*, Advanced Information and Knowledge Processing, Springer London, 2008.

[171] D. B. Knoester, P. K. McKinley, B. E. Beckmann, and C. Ofria, "Directed evolution of communication and cooperation in digital organisms," in *Advances in Artificial Life*, vol. 4648 of *Lecture Notes in Computer Science*, pp. 384–394, Springer Berlin / Heidelberg, 2007.

[172] 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)*, pp. 293–300, 2007.

[173] D. B. Knoester and P. K. McKinley, "Evolution of probabilistic consensus in digital organisms," in *Proceedings of the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 223–232, 2009.

[174] D. B. Knoester and P. K. McKinley, "Evolving virtual fireflies," in *Proceedings of the European Conference on Artificial Life (ECAL)*, 2009.

[175] D. B. Knoester and P. K. McKinley, "Evolution of Synchronization and Desynchronization in Digital OrganismsEvolution of Synchronization and Desynchronization in Digital Organisms," *Artificial Life*, pp. 1–20, 2011. Early access.

[176] 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)*, pp. 137–144, 2009.

[177] J. M. Smith and E. Szathmáry, *The Major Transitions in Evolution*. Oxford University Press, 1995.