This is to certify that the
dissertation entitled

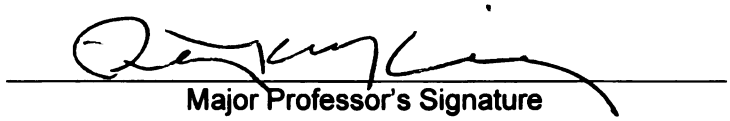
EVOLVING COOPERATIVE, ENERGY-CONSERVING,
AGENT-BASED SYSTEMS

presented by

Benjamin Edward Beckmann

has been accepted towards fulfillment
of the requirements for the

Ph.D. degree in Computer Science



Major Professor's Signature

5/11/10

Date

MSU is an Affirmative Action/Equal Opportunity Employer

EVOLVING COOPERATIVE, ENERGY-CONSERVING, AGENT-BASED SYSTEMS

By

Benjamin Edward Beckmann

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Computer Science

2010

ABSTRACT

EVOLVING COOPERATIVE, ENERGY-CONSERVING, AGENT-BASED SYSTEMS

By

Benjamin Edward Beckmann

Natural organisms are remarkably well-adapted to survive under a myriad of environmental conditions. The robustness and adaptability of natural systems has encouraged research focused on exploiting observed natural phenomena to produce robust, *biologically-inspired* computational systems. For example, biomimetic routing protocols, self-organizing robotic swarms, and artificial immune systems have been successfully developed and deployed. However, these applications are constrained to *observations* of natural systems, and do not take into account the *process* by which those systems evolved. This dissertation focuses on the evolutionary process, and the effect that energy costs have on evolving agent-based systems.

In this dissertation, we introduce an energy model to a digital evolution system, Avida, and illustrate its effects on evolving populations compared to populations that do not pay energy costs. We demonstrate that the inclusion of the energy model produces more predictable outcomes that are consistent with biological theory. We then apply the energy model and evolve individual and group behaviors, culminating in the evolution of energy conserving, density dependent behaviors.

This dissertation serves as an introduction to the evolution of artificial, cooperative, energy-conserving, agent-based systems. As demonstrated throughout this dissertation, populations of digital organisms can evolve many complex collective behaviors, which can then be studied, potentially enhancing our understanding of the biological world and providing robust, emergent behavioral examples that can be applied to multi-agent computational systems.

TABLE OF CONTENTS

List of Tables v

List of Figures vi

1 Introduction 1
- 1.1 Well Adapted Systems . . . 1
- 1.2 Biologically Inspired Systems Design . . . 3
  - 1.2.1 Biomimetics . . . 3
  - 1.2.2 Autonomic Computing . . . 5
  - 1.2.3 Multi-agent Systems . . . 6
  - 1.2.4 Swarm Intelligence . . . 8
  - 1.2.5 Evolutionary Computation . . . 9
  - 1.2.6 Digital Evolution and Artificial Life . . . 11
- 1.3 Toward Evolving Cooperative, Energy-Conserving, Agent-Based Systems . 12
- 1.4 Thesis . . . 14

2 The Avida Energy Model 17
- 2.1 Background . . . 18
- 2.2 Avida Background . . . 21
- 2.3 Avida Overview . . . 22
  - 2.3.1 Avida Operation . . . 22
  - 2.3.2 Self-replication . . . 24
  - 2.3.3 Avida Environment and Scheduling . . . 24
  - 2.3.4 The Avida Energy Model . . . 26
  - 2.3.5 Real-world and Avida environments . . . 28
- 2.4 Experiments and Results . . . 29
  - 2.4.1 Population Size Variation . . . 32
  - 2.4.2 Gestation Variation . . . 34
  - 2.4.3 Behavior . . . 42
- 2.5 Discussion and Conclusions . . . 49

3 Individual Energy Management 52
- 3.1 Evolving Organisms that Sleep . . . 52
  - 3.1.1 Experimental Setup . . . 53
  - 3.1.2 Experimental Results and Discussion . . . 55
  - 3.1.3 Conclusion . . . 62
- 3.2 Cultivating Phototaxis . . . 64
  - 3.2.1 Methods . . . 65
  - 3.2.2 Experimental Results . . . 68
  - 3.2.3 Related Work . . . 73
  - 3.2.4 Conclusions and Future Directions . . . 75

4 Population Energy Management **77**
4.1 Related Work 78
4.2 Demes and Multilevel Selection 79
4.3 Self-regulating population 81
4.3.1 Experimental Extensions and Setup 82
4.3.2 Experimental Results and Analysis 85
4.4 Population Adapting to Environmental Factors 92
4.4.1 Experimental Extensions and Setup 92
4.4.2 Experimental Results and Analysis 96
4.5 Conclusion 103

5 Quorum Sensing and Quenching **105**
5.1 Background 105
5.2 Quorum Sensing in Digital Organisms 109
5.2.1 Avida Extensions 109
5.2.2 Experimental Setup and Results 112
5.3 Quorum Quenching in Digital Organisms 121
5.3.1 Experimental Setup 123
5.3.2 Results 124
5.4 Conclusion and Future Work 132

6 Conclusion **134**

BIBLIOGRAPHY **138**

LIST OF TABLES

2.1 Tasks required by reactions and treatment dependent bonuses. 30

3.1 Rewarded tasks. 54

3.2 Instruction sequence that when executed completes the AND task. 54

3.3 Evolved code that loops until the resource becomes available. 58

3.4 Sample portion of evolved genome using COLLECT-CELL-DATA 69

3.5 Portion of evolved genome exposed to the SENSE3-AND-ROTATE instruction 72

5.1 Deme size comparision 119

LIST OF FIGURES

2.1 Population (bottom) and composition of a digital organism: genome (top left), virtual CPU (top right) with heads pointing to two locations within the genome 23

2.2 Flow chart of the process to bestow a reward on an organism. 26

2.3 Distribution of evolved population size when using metabolic rate-based scheduling. 33

2.4 Distribution of evolved gestation time when an organism can complete a reaction only once. 35

2.5 Average organisms gestation time in each run with no reaction restrictions. Dashed horizontal line at 2000 represents the maximum possible gestation. 37

2.6 Average gestation time over evolutionary time of all 50 runs in the exponential treatment with an inflow rate of 0 and no reaction on restriction completion (left). Black lines represent runs whose average gestation time was low at the end of the run. Lighter colored lines are runs whose gestation was high at the end of the run. Final average gestation of each run with count of runs in each cluster is shown in the scatter plot on the right. 39

2.7 Total reactions performed when each reaction can be performed only once per organism. 44

2.8 Variation of resources that are required for reactions when each reaction can be completed only once per organism. 45

2.9 Total reactions performed when reaction completion restrictions are removed. 46

2.10 Variation of resources that are required for specific reactions when reaction limitation is not present. 47

2.11 Resource equalization cycle with lines that point to resources whose utilization is selected for. 48

2.12 Ecotype diversity of all runs when an organism can perform a reaction only once. 50

2.13 Ecotype diversity of all runs without reaction restrictions. 51

3.1 Comparison of sleep responses in two environments, one where the resource is available 100% of the time (constant), and one where the resource availability decreases over time (declining). Results are the average of 50 runs. 57

3.2 Number of SLEEP instructions present in and executed by organisms in the constant and declining environments. Average over 50 runs. 59

3.3 Representations of a population's response to the resource availability over a single 256 time-step day. Black squares represent sleeping organisms and white squares represent awake organisms. The resource is available for the first 112 time steps. a) t = 1, 231 sleeping, resource becomes available; b) t = 64, 108 sleeping; c) t = 128, 469 sleeping; d) t = 152, 1355 sleeping, resource is no longer available; e) t = 180, 2111 sleeping; f) t = 204, 1502 sleeping, organisms are beginning to wake up; g) t = 228, 667 sleeping; h) t = 256, 189 sleeping, day ends and resource becomes available again. . . . 60

3.4 (a) Attempted resource usage by organisms (resource activity) and resource availability vs. time for a typical 3-day interval. (b) A comparison of SLEEP instructions (squares) to inert NOP-X instructions (circles); solid lines indicate the frequency with which each instruction is found in the genome and dashed lines indicate the frequency at which they are executed. 63

3.5 Model of organism migration starting at the equator where the resource is abundantly available and moving north to regions with less and less resource available for consumption. 64

3.6 Mock daily resource availability over time as an organism's descendants migrate away from the equator. 65

3.7 Depiction of SENSE-CELL-DATA 66

3.8 Average fraction of total allowed time required for a deme to be replicated in both the COLLECT-CELL-DATA and SENSE-CELL-DATA treatments. . . . 69

3.9 Average fraction of organisms in a population that have moved toward, away, and neutrally with respect to the light source in the SENSE-CELL-DATA treatment. 70

3.10 Average fraction of total allowed time required for a deme to be replicated in the SENSE3-AND-ROTATE treatment. 71

3.11 Average fraction of organisms in a population that moved toward, away, and neutrally with respect to the light source in the SENSE3-AND-ROTATE treatment. 72

3.12 Example paths of dominant evolved genomes from the SENSE3-AND-ROTATE treatment, superimposed on the gradient used in the cultivation stage. 73

3.13 Clips from a movie that shows translated code produce by the SENSE3-AND-ROTATE treatment executing on an iRobot Create system. 74

4.1 Population (bottom), sub-population (middle) and composition of a digital organism: genome (top right), virtual CPU (top left) with heads pointing to locations within the genome 80

4.2 Example showing deme initialization and replication of germlines 81

4.3 Example grid containing an organism S, and the cells reached by broadcasting with varying radii. 83

4.4 Deme setup with a nest (0), target (> 0), and empty (-1) cells. 84

4.5 Example path resulting from organism moving back and forth on deme diagonal. 85

4.6 Average fraction of total possible time to complete a deme-level task using multiple energy transfer percentages. Results are mean of 30 runs. 87

4.7 Average fraction of total possible organisms per deme using multiple energy transfer percentages. Results are average of 30 runs. 88

4.8 Average number of organisms in current demes who have performed either of the two individual tasks. Results are average of 30 runs. 89

4.9 Mean of organism gestation times. Results are the average of 30 runs. . . . 90

4.10 Fraction of total possible organisms per deme and fraction of maximum deme gestation time when energy is abundant and 0% or 1% of the parent deme's energy is transfered to the offspring. Results are representative of 30 runs. 91

4.11 Average organism gestation time when energy is abundant and 1% of the parent deme's energy is transferred to the offspring. Results are representative of 30 runs. 91

4.12 Example of the change in execution flow of organism *B* when it receives a high-state alarm from organism *A*. 94

4.13 Example showing organism *A* (represented by a boolean OR gate) before (left) and after (right) executing the ROTATE-TO-ATTACK-CELL instruction. The attack is represented by the lighting bolt. 95

4.14 Mean fraction of total possible organisms within a deme when attacks are present and when they are not. 97

4.15 Average fraction of attacks quelled per attack period in a deme for a single run. 98

4.16 An evolved genome that executes differently depending on if a neighbor has sent a local high-state alarm message. 100

4.17 Sample population experiencing a period of attack. An organism is represented by a boolean OR gate where the point (or output) of the gate denotes the organism's facing. In addition, attacks, quelled attacks, and alarm messages are represented by lighting bolts, crosses, radially blended gray circle, respectively. Underlying each figure is a description of the individual executions represented. 101

4.18 Mean number of births per deme in a single run. 103

5.1 Context switch from sequential execution to an interrupt handler and back. . 111

5.2 Sample genome containing a single interrupt handler. During sequential execution the first four instruction of this genome are executed. If a message is received the organism's IP is jumped into the interrupt handler. This example also contains "junk" code that will not be executed unless the genome is mutated. 112

5.3 Mean fraction of energy remaining per deme over deme generations for each treatment. 115

5.4 Average number of births per deme for all three treatments. 115

5.5 Average organism density per deme. 116

5.6 Organism density and total births per deme for dominate organisms. 117

5.7 Average number of organisms interrupted per deme. 118

5.8 Evolved dominate genome that produces the lowest average organism density. 120

5.9 Average organism density under multiple deme sizes. 121

5.10 Images of a 100×100 deme initially seeded with the genome shown in Figure 5.8. Figure 5.10(a) shows the initial state of the deme with a single uninterrupted organism in the lowest, left-most cell. Figures 5.10(b) and 5.10(c) show a steady exponential increase in population size where the majority of organisms are executing sequentially. Figures 5.10(d) and 5.10(e) depict the rapid behavioral change from self-replication to suppression of self-replication. Lastly, Figure 5.10(f) shows the state of the deme population 40% of the way through a competition period. 122

5.11 Fraction of runs that evolved a quorum mechanism under constant mutant introduction rates. Results are the mean of 20 runs sampled at each of the 25 configurations, marked by a circle. 125

5.12 Fraction of living demes exposed to mutants after a competition period. Results are the mean of 16 runs, one for each dominant exhibiting QS, sampled at each of the 25 configurations, marked by a circle. 127

5.13 Fraction of runs subjected to impaired mutants that perform QS at the end of each stage. Results are a composite of 160 runs for each. 128

5.14 Fraction of organism interrupted per deme during evolution in the staged environment. Error bars denote one standard deviation from mean. Data are a composite of 160 runs. 130

5.15 Mean organism density of dominant genomes extracted from runs exposed to a staged increase in send-impaired mutants. Error bars are omitted for clarity. Data are a composite of 160 samples per line. 131

5.16 Mean organism density of dominant genomes extracted from runs exposed to a staged increase in receive-impaired mutants. Error bars are omitted for clarity. 132

Chapter 1

Introduction

1.1 Well Adapted Systems

Natural organisms are remarkably well adapted to their environment. A single natural organism contains a myriad of individual interacting parts. Each part plays a role in the organism's health and well being. Throughout the evolutionary process these parts have adapted, both morphologically and behaviorally, so that they are well-suited to their environment. Those organisms that exhibit beneficial traits, such as robustness to perturbation, efficient resource usage, and various levels of intelligence, have prospered due to natural selection. Understanding the process by which bio-complexity arose will improve our ability to design and build complex computational systems.

As software developers we strive to create systems that exhibit many traits observed in natural organisms and are as well adapted to a virtual environment, as natural organisms are to their physical environment. Achieving these goals in distributed computing systems typically requires interaction and cooperation among multiple software *agents*, executing on and migrating among nodes in a computer network, including collections of sensors and robotic systems. Effective cooperative behavior in such settings must overcome adverse conditions, including nodes that are malfunctioning or physically compromised.

To assist in manifesting these properties/characteristics within a distributed system, we augment software development with evolutionary methods, specifically *digital evolution* (DE). Through DE, a developer can explore alternative solutions by mapping a real-world system's capabilities and intended environment into an evolutionary setting, enabling an evolutionary process to stochastically search for fit solutions. The solutions produced through this process can be studied for novel behaviors that optimize resource usage.

There are many examples of naturally occurring collective behaviors that optimize resource usage. Social insects commonly act as a group to perform a task. For example, leafcutter ants form "highways" to transport portions of leaves, harvested from plants and trees, to their nest [87]. Other social insects, such as the wasp *Polybia occidentalis*, divide nest-building duties among three groups of workers (pulp foragers, water foragers, and builders), and regulate the size of each group by exchanging information [95]. Observations of these and other group-level behaviors have been widely studied in biological circles [87, 132, 192] and interest in them can be traced back as far as the 9^{th} Century [80]. In 1975, Wilson introduced the field of Sociobiology [192], which links these naturally occurring social behaviors to the evolutionary process, fundamentally underpinning the research described in this thesis.

Collective computing systems, such as wireless sensor networks and robotic swarms, must collectively optimize resource usage. They are susceptible to performance degradations caused by uncontrollable environmental features, such as unpredictable network disruptions. As these systems increasingly interact with the physical world they are often required to operate in extreme situations. To remain effective, these systems must adapt to dynamic network conditions, conserve energy, compensate for hardware and software failures, fend off attacks, and optimize performance, all with minimal human intervention [102, 127]. Moreover, achieving these goals typically requires interaction and cooperation among multiple nodes, some of which may be malfunctioning or physically compromised. The complexity of designing robust computational systems has led many

researchers to investigate *biologically-inspired* approaches. Let us briefly review these methods.

1.2 Biologically Inspired Systems Design

Recently, biologically-inspired methods for designing software systems [1, 4, 9, 32, 40, 77, 96, 105, 121, 122, 153, 154, 174] have gained popularity in research communities. Examples include biomimetics [98], autonomic computing [102, 187], multi-agent systems [172], swarm intelligence [24], and evolutionary computation [39, 45, 71, 84, 107]. This push toward biologically-inspired approaches for computational system design stems from observations of robustness in natural organisms when they are confronted with dynamic conditions.

1.2.1 Biomimetics

The morphologies and behaviors of individual organisms are exceptionally well suited for their environments. For example, a hummingbird's ability to hover enables it to feed on the nectar of plants, and a cheetah's agility and speed allow it to catch other less capable animals. In addition, *collaborative* behaviors are also present in nature in many forms and at many levels of complexity. For example, many bacteria perform quorum sensing [132], enabling the completion of collaborative tasks such as the formation of biofilms [78, 165], swarming motility [165], exopolysaccharide production [165], and cell aggregation [165]. Social insects also collaborate to construct intricate nests, survive attacks, travel long distances, collect food, and perform many other tasks [30]. And of course, cooperation is pervasive in many other species, including human beings. Similarly, complex cooperative behaviors are also required in computational systems. Robotic swarms [56, 99, 100, 180], intrusion detectors [139], and collective energy management in sensor networks [109] all require complex cooperative behaviors.

The ability of natural organisms to perform collaborative tasks while adapting to unforeseen circumstances has led researchers to study biomimetics [98]. An approach is considered to be biomimetic if it mimics natural occurrences within a man-made entity. Many biomimetic approaches have been proposed [4, 9, 32, 40, 122, 174], including biologically-inspired software design patterns [9]. One of the patterns suggested in [9] is stigmergic communication (SC). SC is a form of indirect communication where an individual places information into its local environment. The placement of this information can affect a decision made by another individual in that location sometime in the future. This method of communication, inspired by species of ants that use pheromones to communicate the location of food, a nest, and so on [87], has been used in routing algorithms to efficiently direct packets to a destination [32,51]. An example of the SC pattern can be found in [122], which presents the "tuples on the air" (TOTA) middleware. In TOTA, an agent can communicate indirectly through its environment by propagating and sensing locally available tuples, thereby enabling SC among agents.

In addition to the design of software systems, biomimetic approaches have also been used to design physical objects, such as robots [7, 89, 118]. As stated in [7], "Biomimetic robots differ from traditional robots in that they are agile, relatively cheap, and able to deal with real-world environments." Biomimetic robots have been developed for aquatic [8, 175], terrestrial [88, 108, 164], and aerial [195] domains with various levels of complexity and ability. Extending these ideas, the field of evolutionary robotics [144], described in Section 1.2.5, combines both biomimetic software and hardware with an evolutionary process, potentially sidestepping incorrect engineering constraints or preconceptions and allowing for "uncontrolled engineering" [117].

Biomimetic software and hardware have been applied to address many cyber-related problem domains, as described above. However, as the complexity of these and other computing systems continues to increase, both in scale and functionality, traditional methods of control requiring a "human-in-the-loop" will be inadequate. Moreover, as computer

systems increasingly interact with the physical world, they will require a higher degree of autonomy. To aid in the management of increasingly complex systems, additional control mechanisms that limit or remove the human control component, will be required. Researchers have proposed a subfield of biomimetics, called *autonomic computing* [94], to fulfill this management need by enabling a system to govern itself.

1.2.2 Autonomic Computing

Many species of plants and animals have evolved self-governing systems that are critically important to their survival. An example in humans is the autonomic nervous system, which regulates our respiration rate, digestion, heart rate, perspiration, pupil dilation, salivation, micturition, and sexual arousal. Leveraging this knowledge and anticipating the management needs of future computing systems, IBM published a manifesto detailing the need for self-governing, or autonomic, computing systems [94]. In addition, Kephart and Chess promoted IBM's vision of autonomic computing in [102], supported by a computational architecture in [187]. They suggested to focus on the creation of computational systems capable of self-management, according to the goals of the system's administrator, in order to reduce the burden of system management. In doing so, they sparked numerous research efforts relating to autonomic computing in government [180], industry [120, 173], and academia [52, 56, 128, 129, 194].

According to Kephart and Chess [102], a system can be called autonomic if it is capable of self-configuration, self-optimization, self-healing, or self-protection. The goal of an autonomic system developer is to create a system that, after it is started, requires little or no human interaction. To create such a system, many concepts (such as control theory [103], adaptation [127], and domain knowledge) must be combined into a single coherent system. Furthermore, the developer of an autonomic system must ensure the system's quality of service (QoS) will degrade gracefully in the presence of adverse conditions, and that its responses will not negatively affect other interfacing systems. Indeed, interaction among

multiple entities in meeting high level system objectives is a major aspect of autonomic systems.

1.2.3 Multi-agent Systems

As computing becomes more pervasive and decentralized, many design techniques, including autonomic agent-based systems [102], have been proposed to accommodate the increasing complexity. An agent is typically defined as an independent decision maker residing within an environment. Each agent is capable of local sensing and local communication, and is solely responsible for its computational load and resource usage. Autonomous agents have been applied to many different application areas in computer science [11, 23, 102], and the agent paradigm has been shown to be an effective approach to addressing dynamic reconfiguration [67, 172, 188], scalability [66, 172], and self-protection [172]. These advantages are derived from an agent's ability to act either individually or collaboratively, to clone itself, and to migrate across networks while maintaining state and performing tasks. An agent's ability to be either active or passive depending on the state of the system is another major advantage of agent-based systems. For example, when a system is not experiencing events of interest, an agent can remain in a quiescent state, periodically waking and sensing the local environment. Once an interesting event is detected, the detecting agent can become active and send messages to other active agents, as well as spawn clones at neighboring nodes in order to monitor an event [67]. Systems capable of such functionality are typically referred to as *multi-agent systems* (MAS) [172]. Similar to the delegation of roles within a nest of honey bees [30], a MAS comprises many, possibly heterogenous, agents collaborating to perform a task. Many biomimetic systems (e.g. an artificial immune system [68, 83]) require multiple individuals interacting in a coordinated fashion.

A MAS may be preferred to a single monolithic agent for many situations [99, 172]. The inherent parallelism in a MAS provides a suitable method of distributing the system's workload. Also MASs, as described in this thesis, do not have a single point of failure,

so they are robust to agent-level faults. Scalability is also a feature of a MAS because of the inherent modularity of an agent: if more agents are required to perform a task, they can be easily created. In addition, the modularity of an agent increases the system's adaptability, in that new agents can be added as new capabilities are required. A MAS also allows a programmer to subdivide a problem into simpler tasks instead of tackling the entire problem at once; an individual agent can be designed/assigned to a specific subtask. Physical distribution of agents in a MAS is also an advantage since it enables shorter system response times than a single agent.

Many computational systems have been designed using the MAS paradigm; examples include wireless sensor networks [66, 67], robotic swarms [55, 56], and artificial immune systems [68, 83]. In these systems each node or robot contains an agent that is responsible for its operation. In addition, the agent collaborates with other agents in its vicinity to help perform a task. Combined, the system of agents is capable of performing tasks that a single agent cannot [52, 56, 58, 66, 168].

In addition to benefitting the design of distributed systems, MASs are also useful for studying fundamental problems in life sciences [61, 150, 158] and cognitive sciences [31]. For example, Epstein and Axtell have show macro-behaviors arise in resource constrained environments [61], and Ray has demonstrated parasitic behavior in an evolving MAS. In this thesis, we focus on developing cooperative MASs through the use of evolution, and study emerging group-level behaviors that arise from interactions among individuals. As described in [60] and [186], "intelligence" does not reside within an agent, but between agents and is viewable through interaction. Therefore, as proposed in [41], the best way to develop intelligent machines may be to start with "social" machines. Following this suggestion, we explore interactions between agents, and between agents and their environment (collectively, their observable behavior) to determine the success or failure of a task. This concept is closely related to swarm intelligence.

1.2.4 Swarm Intelligence

The field of swarm intelligence (SI) [24, 100] focuses mainly on the end-product of sociobiology, specifically leveraging the social insect metaphor to build efficient, flexible, and robust computational systems. Swarm intelligence as defined by Bonabeau *et al.* in [24] is "any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies." This broad definition lends itself well to many SI subfields, such as ant colony optimization (ACO) [53,54], and in addition does not exclude the processes by which these collective behaviors came about, namely, natural selection. Fundamentally, SI can be thought of as applying the behaviors encompassed by sociobiology to problem solving.

SI focuses mainly on producing group behavior through self-organization [24, 100]. A system is considered to be *self-organizing* if interaction between individuals lead to an emergent, complex global behavior. For example, individual wasps perform foraging and building tasks to collectively construct a nest. Inherently, a self-organizing behavior is a bottom-up phenomenon, and achieved through multiple methods in both natural [30,87,98, 192] and artificial [4,57,64,98,123,167,174] systems.

In computational settings, SI researchers have demonstrated a variety of collective behaviors among homogeneous groups of robots, including aggregation [13, 177], coordinated motion [12, 55], collective and cooperative transport [74–76], adaptive task allocation [110–112], navigation on rough terrain [178], and functional self-assembly [179]. Collective behaviors for a heterogenous group of robots have also be successfully explored [181]. We extend this research and focus on evolving energy efficient cooperative behaviors, while enhancing the understanding of the evolutionary process. To do so we will need to employ methods from the field of evolutionary computation.

1.2.5 Evolutionary Computation

As noted earlier, *biomimetic* approaches to designing robust computer systems [98] are approaches in which successful natural behaviors are mimicked *in silico*. However, while purely biomimetic methods are effective in certain domains, they are constrained to observations of nature, and do not take into account the evolutionary process that created our amazingly complex global ecosystem. Furthermore, natural organisms are well-adapted to their environment because they have been exposed to both structural and behavioral optimizations through the evolutionary process. Being well suited to its environment is also a concern for a computational system. Therefore, applying methods similar to those that produced efficient resource usage, instincts, cooperation, intelligence, and creativity in the natural world is a logical choice to investigate and apply to the design of agents in a virtual world.

Evolutionary computation (EC) [45] is a broad field focused on applying the basic principles of natural selection to solving optimization problems. EC includes any computational system that satisfies the three conditions necessary for evolution to occur [49]: replication, variation (mutation), and differential fitness (competition). The most well-known EC method is the genetic algorithm (GA) [84], an iterative search technique in which individuals in a population are encodings of candidate solutions. In each generation, the fitness of every individual is calculated, and a subset is selected, recombined and/or mutated, and moved to the next generation. Genetic programming (GP) [71, 107] is a related method where the individuals are often actual computer programs. Both approaches have been successfully used to solve complex problems in distributed computing, such as multicast mapping [77], network intrusion detection [1], process scheduling [121], topology optimization and routing [105, 153], and peer-to-peer protocols [40]. In some cases, EC methods have produced solutions that are competitive with state-of-the-art human design and patentable innovations [107].

One common use of a GA is to train the "brain" of an agent through neuro-evolution

[65, 131, 144]. In this approach, the GA "evolves" the neural network by modifying the connections between the network's nodes. Over evolutionary time the GA evaluates solutions using a fitness metric that provides selective pressures to guide the evolutionary process toward solutions that exhibit a desired behavior. Other neuro-evolution methods enable the structure of the neural network to be modified as well [170, 171].

In addition to evolving a single agent's behavior, EC methods have also been used to evolve collective behaviors within a group of homogeneous [39] and heterogeneous agents [27, 197]. For example, Yong and Miikkulainen [197] extended prior work on evolving individual neurons within a neural network [73, 137, 138] to a multi-agent setting. In [197], the controller for each agent within a group is evolved separately, allowing each agent to specialize its controller to a specific task, producing heterogenous agent controllers and behavior. However, the group of agents is evaluated as a whole to determine the fitness of the evolved controllers, and credit for this fitness is evenly assigned to each participating controller.

Another method described in [39] focuses on the use of a developmental model, where the controller of each agent in a group is produced by the same controller generator. The generated controller's behavior is dependent on the agent's context, therefore the evolved controller generator produce a set of controllers with varying behaviors. In short, the method described in [39] produces a single solution that, depending on context, can "unroll" into different agent behavior. Other recent work focusing on evolving homogenous control structures for robots with heterogenous sensing capabilities has also shown promising results [181].

The study of embodied robotic agents in the EC field is called evolutionary robotics (ER) [93, 144]. ER focuses on both the application of EC methods to the design of robotic controllers [93, 144], and in some studies the co-evolution of a robot's controller and morphology [119, 169]. For eample, work on the Swarmanoid [52] project addresses the "design, implementation and control of a novel distributed [heterogeneous] robotic system,"

extending its predecessor, the Swarm-bots project [56], which used homogeneous robots.

In addition to the application of EC to solve optimization problems, other works focus on enhancing knowledge of the evolutionary process [90, 130]. For example, Holland created the ECHO system [86] to study emergent behavior of a population of agents. The ECHO system, much like the Avida system described in the next section, provides researchers with the ability to perform virtual experiments to study the dynamics of the evolutionary process. Over the course of evolution, agents develop strategies that help ensure their survival in a resource-limited environment.

1.2.6 Digital Evolution and Artificial Life

In this work we apply evolutionary computation to the development and analysis of individual and group behavior. Instead of using a genetic algorithm (GA), however, we explore the potential benefits of another form of evolutionary computation known as *digital evolution* (DE) [2], which is closely related to several artificial life platforms [2, 85, 114, 157, 160]. In digital evolution, a population of self-replicating computer programs exists in a computational environment and is subject to mutations and natural selection. These "digital organisms" are provided with limited resources whose use must be carefully balanced if they are to survive. Since digital organisms interact with one another and are responsible for their own replication, the process is closer to natural evolution than traditional forms of evolutionary computation such as GAs. Designed primarily as a tool for evolutionary biologists, this "digital Petri dish" enables researchers to explore questions that are difficult or impossible to study with organic life forms [3, 35, 116, 189, 190]. For example, Lenski *et al.* used Avida [149] to study the evolutionary origin of functional artifacts that may appear too complex to have been produced through evolution by natural selection [116]. This article and others have demonstrated that fundamental principles of evolution can be observed using digital evolution [2, 35, 113, 114, 151, 157, 158, 190].

In addition to uses in biology, digital evolution provides a means to *harness* the power of

natural selection and apply it to a variety of problems in designing computational systems. Digital evolution enables the designer to explore an enormous solution space, potentially discovering unintuitive algorithms that are robust under highly dynamic and adverse conditions. The investigations described herein explore the application of digital evolution to problems in distributed computing [126].

However, Alife systems, such as Avida, represent an abstraction of natural phenomena. They include only features that are built in, and exclude others that *may* be important to furthering our understanding of the evolutionary process. At a minimum, any evolving system must include replication with variation and differential fitness (competition) among individuals, as these are essential ingredients of the evolutionary process [49]. Building on this recipe, evolving systems can be constructed to test hypotheses and potentially contribute to the formulation of evolutionary principles. However, as with cooking, the quality of the ingredients will affect the result. Alife system designers must pay close attention to the realization of these required ingredients in their evolving systems, as they can affect the outcome.

1.3 Toward Evolving Cooperative, Energy-Conserving, Agent-Based Systems

As we have seen, cooperative agent-based systems are common in both natural and human-designed systems. In addition, they have many intrinsic benefits that can be leveraged to build a distributed, autonomous applications. Furthermore, virtual agent-based systems can also be used to test hypotheses about natural phenomena and learn about self-organizing emergent behavior. However, very little work has focused on the effects of energy on the evolution of an agent-based systems. In this dissertation, we explore the evolution of cooperative, energy-conserving, agent-based systems.

Energy efficiency can be used both directly and indirectly to influence the evolutionary

process. If selection is based solely on how well an individual completes a task, and energy efficiency is ignored, then solutions can evolve to be successful, but their translation into real devices can produce suboptimal results. For example, as shown in [125] when a robot was evolved to search for an object without any energy constraints, it evolved to spiral out from its starting position and ignored possibly useful sensor readings. In contrast, when energy was included in the fitness evaluation the robot actively polled its sensors and chose a more direct path to the target, increasing its energy efficiency. Floreano *et al.* [65] also used energy efficiency indirectly to evolve a movement strategy for a mobile robot. The robot was evolved to move for an extended period of time using a rechargeable battery. As long as the robot returned to the recharging sector of the arena before its battery was depleted, its battery would be recharged and it could continue to move, increasing its fitness. Through the evolutionary process the robot evolved to recharge itself and survive until it was stopped due to a hard time limit.

These direct and indirect uses of energy exemplify the theory of diminishing returns, which refers to the point at which additional resources translate into a negligible return on investment. Ideally, a computer system should automatically operate close to this point, optimizing its performance. Commonly, the configuration required to achieve such performance, such as the number of agents within a systems, is determined *a priori* as in artificial immune systems [83] and particle swarm optimization [156]. Our focus here is on the effects energy efficiency can have on the evolution of group-level tasks, specifically the number of agents involved.

Our studies show that energy is integral to the evolutionary process. Specifically, when energy is considered as a consumable resource, selective pressures drive evolution toward energy conserving strategies. Leveraging this insight, we study evolved group behaviors, looking for collections of individual behaviors that when combined produce emergent group behaviors. Providing better understanding of these individual and group behaviors will aid in the design of robust MASs by mechanisms that have been successfully used to

produce effective group behaviors in silico.

1.4 Thesis

This dissertation focuses on implications of incorporating energy into Avida. Traditionally, in many Alife systems [149, 151, 157, 158], including Avida, an individual is allowed to execute whenever it is allocated time by a scheduler. However, this model does not account for the fact that performing a behavior requires an organism to pay both time *and energy* costs. The inclusion of an energy model, described in Section 2.3.4, requires that an individual pay an energy cost in addition to a time cost. This dissertation will show that the addition of energy costs can significantly change the evolutionary trajectory taken by a population, compared to those that are not subjected to these "extra" costs. Specifically, this dissertation will demonstrate that:

Thesis Statement: *An individual's function requires both time and energy, and the optimization of these resources produces phenomena that are desirable in multi-agent systems.*

This dissertation provides direct evidence that incorporation of an energy producing metabolism alters the results of an evolutionary experiment, contributes an implementation of the energy model described later, illustrates the how the energy model can be used to evolve individual and group behavior, and proposes an implementation of active messages that trigger threaded message handlers. Specifically, the dissertation will contribute the following.

1. We explore the differences produced within and between evolutionary trials where individuals pay extra energy costs and where they do not. Organism-level energy will be introduced as a commodity that must be well managed for an organism to survive. The introduction of energy produces results consistent with evolutionary theory and avoids divergent evolution that is present in other non-energy trials.

2. We explore the issue of energy efficiency within an individual. We describe a set of experiments that demonstrate the ability of the evolutionary process to produce resource-aware adaptive behavior (i.e., sleep). In addition, we study the effect of a diminishing resource on the evolution of energy-efficient organisms. Lastly, we demonstrate the evolution of energy-conserving phototaxis. These experiments are initial steps toward evolving energy-efficient agents for deployment on resource-constrained devices.

3. We investigate the role group-level energy efficiency can play in natural selection, in particular, its effects on the self-regulation of a group's population. For example, when a group is selected for replication, what happens if its previous energy gains are inherited completely, partially, or not at all? What effects does group-level efficiency have on the number of individuals required to complete the task and their behavior? Does energy abundance increase or decrease the time required to evolve a group-level task? In addition to providing evidence that helps to answer these questions, we also apply the results to the design of agent-based distributed systems.

4. We demonstrate a "proof of concept" that DE can produce a population of mobile agents exhibiting self-organization, specifically, cooperative actions among neighboring organisms to mitigate attacks, and self-adaptation, by changing their collective behavior in response to dynamic threat levels. This study also helps to identify evolutionary conditions that enable this global behavior to emerge in a population. Moreover, analyzing the genome of a particularly successful population provides insight into the complexity and effectiveness of solutions discovered through the evolutionary process.

5. We demonstrate that density-dependent behavior can be evolved using basic energy conservation pressures. We introduce to Avida active messaging capabilities that enable one agent to affect the execution of another. Additionally, we show that evolved

density-dependent group behaviors are resistant to specific impairments and evolve in the presence of impairments.

This dissertation is organized into five remaining chapters. Chapter 2 describes the Avida platform [149] used in this study, motivates the need for this research, introduces the energy model, and demonstrates that the inclusion of an energy-producing metabolism can alter the outcome of the evolutionary process. Chapter 3 describes initial experiments in which organisms evolved to conserve energy. Chapter 4 extends the evolution of energy conservation from a single organism to a group of organisms. Specifically, groups of organisms are evolved to cooperate to perform a group-level task while conserving energy. Chapter 5 extends these results to an example group behavior where organisms are evolved to sense a quorum, and assess the effects of attempts to disrupt this behavior. Finally, Chapter 6 discusses areas of possible future investigations beyond this dissertation.

Chapter 2

The Avida Energy Model

In this chapter we focus on implications of incorporating energy into an Alife system. For this we extended the Avida digital evolution platform [149], described in Section 2.3. Traditionally in Avida, and many other Alife systems [151, 157, 158], an individual is allowed to execute whenever it is scheduled. The inclusion of the energy model, described in Section 2.3.4, requires that an individual pay an additional energy cost that can significantly change the evolutionary trajectory taken by a population.

In this work we demonstrate, across a wide-range of environments, the benefits and limitations of incorporating energy costs into an Alife system. In Section 2.4 we demonstrate that the inclusion of energy allows for the control of a population's size through the inflow of resources, which contrasts with many systems where the population's size is constrained only by available space [151, 157, 158]. Moreover, when the size of a population is limited by its energy, as its constituents evolve to collect more energy, and more efficiently, the size of the population can also evolve. We also show that the inclusion of energy enables configurations where a population's limiting resource can be physical space or inflowing resources.

2.1 Background

Many "traditional" evolutionary algorithms, where an individual represents a complete solution to an optimization problem (such as a genetic algorithm [84] or genetic program [106]) evaluate individuals atomically and synchronously [45]. Lock-step evaluation of individuals is acceptable since individuals do not interact. However, lock-step execution and/or evaluation of individuals does not accurately model the asynchrony inherent in a natural system. Rather, in a natural system, where genetic material is passed vertically, future generations depend on whether an individual survives long enough to reproduce. Furthermore, the survival of an individual is not a solitary endeavor. An individual's survival can be affected by the behavior of others and by the environment. We argue that to preserve the asynchrony among individuals, which affects survivability and thus evolution, an Alife system designer must be conscious of three key features: the scheduling of individuals, the environmentally implied fitness function, and time and energy costs paid by individuals. These three features are critical to competition, which is an essential ingredient of the evolutionary process.

Scheduling of Individuals. The modeling of asynchronous features in an Alife system can be reduced to a scheduling problem. When does an individual's state change? Depending on the model, the answer to this question varies. For example, in Conway's Game of Life, popularized in [69], the state of every cell is evaluated each generation. Therefore the state of all "individuals" can change simultaneously and their interactions occur in lock-step. Likewise, updating of state in Coreworld [157] also happens in lock-step. However, there is no built-in concept of an individual. Instead, the state of core memory and instruction pointers are updated in parallel. A third example, where the genetic code of multiple individuals may overlap, is Tierra [158]. In Tierra an individual consists of a virtual CPU whose state is changed when its instruction pointer reaches the top of the "slicer queue" and is then executed. In this way, individuals are scheduled sequentially. A final example

is Avida, the experimental system used in this study. In Avida an individual consists of its own genetic material and a virtual CPU that executes its genetic material. Avida probabilistically schedules an individual, based on the performance of its ancestors, to determine when its state changes. Individuals with highly competitive ancestors are probabilistically scheduled more often than those whose ancestors were less competitive. This scheduler allows individuals to execute asynchronously. Specifics of the scheduling methods used in the following experiments will be presented in Section 2.3.

Implied fitness. The success of a natural organism depends on organismal and environmental interactions. An organism's behavior and physiology along with resource availability and other discernible features play a role in its success as a replicator [43]. In natural systems there is no omniscient fitness function evaluator that decides whose genes are to be passed along to the next generation and whose are not. Instead, an organism's ability to pass along its genes is implied thought its extended phenotype [44]. The organism must be able to reproduce, collect and metabolize food, and perform other aspects that are essential for life. Therefore an organism's success is implied by what is does, who it meets, where it lives, when and why it performs specific tasks, and how efficient it is at completing these tasks. Entangled in these interactions are costs and benefits that provide the organisms with a "merit" and shape how the organism's lineage evolves.

Interactions among individuals are dynamic in an evolving system, whether the system is natural or artificial. For example, an individual's consumption of resources may depend on the rate of consumption by others. Moreover, an individual's successful replication may be inhibited by others who replicate more quickly. Even if the environmental conditions are constant, as evolution progresses the interactions among individuals change as the population adapts and more successful individuals evolve. In a simple environment where individuals only compete for space (*e.g.* other resources play no role in competition), those that replicate more quickly will be successful, which will drive additional innovations lead-

ing to shorter gestation cycles. Therefore, even in a "static" environment, the interactions among individuals is dynamic, and the success of any individual will depend on its ability to handle such conditions.

Time and Energy Costs. Many artificial systems reward an individual for performing user-defined tasks [61, 86, 149]. These rewards partially or fully determine whether an individual's genes are passed on to the next generation or not. However, this type of system does not account for how efficiently an individual completes a task if all of the individuals are evaluated simultaneously. Many Alife systems overcome this hurdle by modeling an individual's execution over time [61, 85, 86, 149, 157, 158]. In these models, an individual's lifetime is extended from a single time step, as in the Game of Life, to many time steps, the sum of which is the individual's gestation time. Gestation time enables the selection of genetic structures that produce faster replicators. Moreover, in systems with self-replication, the lifetime of an individual can vary [149, 151, 157, 158], which promotes competition. The inclusion of time costs for completing tasks allows for selective pressures that reward efficiency, at least with respect to time. Individuals that perform tasks more quickly have an advantage over slower competitors. For example, in Tierra [158] individuals are responsible for their own self-replication and compete for space in memory. Total memory size in Tierra is fixed, therefore an evolutionary pressure arises that drives a decrease in average gestation time. Interestingly, using this model, complex parasitic interactions have been evolved where individuals trick others into replicating their genetic code. This trickery reduces the amount of time required to replicate the trickster. [159].

To summarize, every Alife system has one basic feature in common, scheduling. How individuals get scheduled for execution greatly affects the results produced by a system. Individuals can be scheduled in parallel or sequentially. Individuals may be prioritized based on some metric, allowing those who have achieved a higher level of reward more time to prosper. However, one scheduling-related feature that is pervasive in natural organisms but

underrepresented in Alife systems is *energy*. Natural organisms must metabolize resources to gain energy and sustain life. Modeling of this processes, specifically the extraction of energy from the environment and its use to perform tasks, provides another dimension by which selection can act. In combination, time and energy costs may alter the evolutionary trajectories that are taken by an evolving system. The experiments in Section 2.4 demonstrate the evolutionary effect of including energy costs, and illuminate areas that require more investigation.

2.2 Avida Background

While evolutionary computation has been studied since the 1960's, the specific field of digital evolution is much younger. The first experiments with populations of self-replicating computer programs were performed by Rasmussen in a system that he dubbed Coreworld [157]. However, this system was relatively brittle and soon thereafter Ray designed Tierra [158], which used a streamlined and fault-tolerant genetic language. In 1993, Ofria and colleagues began development of Avida [149, 150], currently the most widely used digital evolution platform and the one used in our research.

Avida was developed primarily to provide a better understanding of evolution in nature. Observing evolution in digital organisms enables scientists to address questions that are impossible to study with organic life forms. The Avida platform has been used to conduct pioneering research in the evolution of biocomplexity, with an emphasis on understanding the evolutionary design process. Specific studies address the evolutionary origin of complex traits [116], the evolutionary design of modularity [134, 135, 145] and robustness [48, 115, 190], the evolution of multiple, interacting programs [35, 36, 70, 148], the design of evolvable programming languages [146], and analysis techniques to break down the information contained within evolved code [3, 91, 147].

In addition to providing a better understanding of biological processes, digital evolution

can also help solve problems in science and engineering [35, 146] and to design computing systems that interact with the physical world [126]. Recently, Avida has been used to cultivate digital organisms that exhibit self-organizing [18,104], self-healing [104], and adaptive resource-aware behavior [19], as well as automated state-diagram construction [72] used in software requirements engineering. Our investigations focus on how the harnessing of DE can contribute to the design or synthesis of robust, cooperative, energy-efficient multi-agent systems [126].

In Avida, individuals, or "digital organisms," self-replicate and evolve to perform tasks in a user-defined computational environment. Instead of a traditional fitness-based selection process, in DE an organism's ability to self-replicate drives natural selection. This method of selection more closely matches that of the natural world and can provide insight into the evolutionary process [116], as described above. When applied to engineering problems, digital evolution provides a powerful search tool, as do GA and GP. However, self-replication and the absence of a predefined fitness function also produce evolution that is less constrained by a predefined fitness function. While evolved solutions may share the inherent imperfections of natural organisms, they might also be resilient to some unexpected conditions to which human-designed algorithms are limited and/or brittle.

2.3 Avida Overview

2.3.1 Avida Operation

In Avida, digital organisms compete for space within a fixed-size two-dimensional collection of *cells*. Each cell can contain at most one organism, which comprises a circular list of instructions (its genome) and a virtual CPU that executes those instructions, as shown at the top of Figure 2.1. Instructions perform simple arithmetic operations (addition, bit-shift, increment, etc.), control execution flow, aid in self-replication, enable the organism to move from one cell to another, and provide a means for the organism to interact with other

organisms and the environment. Additional instructions can easily be added as needed. An organism executes instructions on its virtual CPU, which contains three general-purpose registers (AX, BX, CX), two general-purpose stacks, and special-purpose heads that point to locations within the organism's genome. Similar to a traditional program counter and stack pointer, heads are used to control the flow of execution and to facilitate replication.

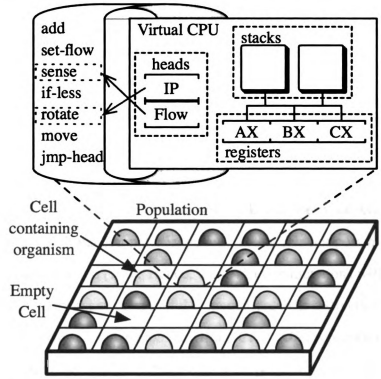


Figure 2.1: Population (bottom) and composition of a digital organism: genome (top left), virtual CPU (top right) with heads pointing to two locations within the genome

The execution of an organism's genome controls its behavior, including replication of its offspring. Instruction execution costs the organism both virtual CPU cycles and energy. The number of virtual cycles and energy expended depends on the instruction being executed. For example, executing an instruction that activates a virtual motor consumes more energy than executing an arithmetic instruction that manipulates registers. This simple von Neumann architecture does limit how efficiently an organism can perform a behavior, but it is suitable for many of our studies. In Section 5.2.1, we propose future work on investigating the integration of an interrupt architecture into Avida.

2.3.2 Self-replication

Self-replication is a critical aspect of digital evolution, and distinguishes it from other forms of evolutionary computation. To avoid extinction, an organism must be able to produce offspring. This means that the instructions comprising the organism's genome must contain a sequence that will create a copy of the genome and split it off into an offspring organism. Initially, an ancestral organism containing a genome capable only of self-replication is injected into the population. In Avida, self-replication requires the organism to copy its own instructions one-by-one to a newly allocated space and then divide the newly allocated genome from its ancestral genome. Each copy operation is subject to random mutations that can change the instruction. In addition, mutations can cause instructions to be inserted or deleted upon division. All behavior other than replication must enter the genome through a series of individual mutations over generations. A key property of Avida's instruction set that differs from traditional computer languages, is that it is not possible to construct a *syntactically incorrect* genome [146]. Hence, while random mutations will produce many genomes that do not perform any meaningful computation, their instruction sequences will still be valid, that is to say, executable on the virtual CPU. After replication, the offspring, possibly containing mutations, is placed in a cell in the population, terminating any organism already residing there. Each Avida run begins with a single self-replicating organism, which is an ancestor to every organism produced during the run.

2.3.3 Avida Environment and Scheduling

The environment in which an organism lives shapes its evolution. For example, as will be shown in Section 2.4.2, when space is the limiting resource the organisms evolve shorter gestation times, enabling them to better compete for the limited space resource. However, organisms can also evolve to perform *reactions* defined by the users. When an organism completes a reaction it receives a bonus that has an effect on the scheduling of its descendants. The size of the bonus depends on the availability of an associated resource.

Completing a reaction that has an abundant associated resource will confer a larger bonus than completing the same reaction when its required resource is depleted.

In Avida a reaction comprises a task, a resource, and the type and size of a bonus received by the organism when it completes the reaction. Tasks in Avida are binary logic functions that an organism must complete to become eligible for a reward. They are defined in terms of an organism's observable behavior, or *phenotype*, which manifests itself in three ways: I/O between an Avidian and the environment, messages sent between two Avidians, and an Avidian's movement. Figure 2.2 shows a flow chart of the process of conferring a reward to an organism. Initially, when an organism outputs a bit-string by executing the `IO` instruction, its "behavior" is tested to determine if it has completed a user-defined task. If a task was completed, then its associated reaction is tested, otherwise the process exits. In order for a reaction to be triggered by the completion of a task, other prerequisites must be satisfied as well. Specifically, the resource associated with the reaction must be available above a minimum level. Additionally, a limit on the number of times a reaction can be performed may also be configured. If a reaction is completed then the organism receives a reward, which influences the scheduling of an organism.

Avida has two mutually exclusive configuration methods for establishing the scheduling priority of an organism. Traditionally, merit-based scheduling has been used. However, we have extended Avida with an energy-based scheduling method described in Section 2.3.4.

When using merit-based scheduling, an organism with a higher merit is scheduled more often than one with lower merit. An organism's merit is established at birth and remains constant throughout its lifetime. If a parent gains merit during its lifetime, its children's merit will be higher than the parent's. Likewise, if a parent achieves a net loss of merit during its lifetime, the result is a lower merit for the children. The completion of reactions by the parent organism directly affects its offspring's merit.

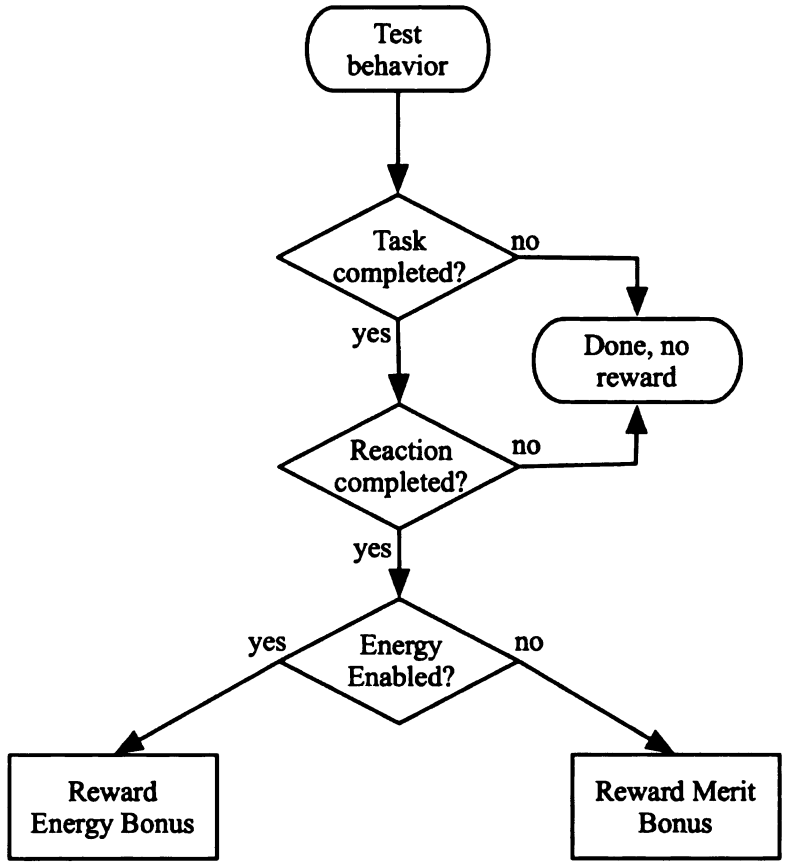


Figure 2.2: Flow chart of the process to bestow a reward on an organism.

2.3.4 The Avida Energy Model

We have extended Avida with an alternative scheduling methodology that requires an organism to pay both time costs and energy costs to execute an instruction. We call this methodology the *energy model* and have used it in various studies [16, 17, 19, 20]. When the energy model is enabled, every digital organism is provided with an energy store. An organism pays to execute instructions with energy from its energy store. When an organism depletes its energy, it dies. An organism receives an energy bonus when it completes a reaction. To remain consistent with the standard Avida configuration, all energy bonuses in the experiments described in Section 2.4 are applied when an organism replicates. They do not affect the organism's scheduling during its lifetime. Upon replication the parent's existing energy store decays, at a user defined rate, and is divided equally among offspring. Disproportional distribution of energy among offspring is configurable. In the experiments

described later the decay is 5%. The decay is intended to model energy lost during self-replication.

Metabolic Rate. In addition to ensuring an organism has sufficient energy to execute instructions, the energy model also affects the scheduling of organisms. The amount of energy in an organism's energy store determines its *metabolic rate*, using Equation 2.1. (In Avida, merit and metabolic rate are analogous within the scheduler, however we will use the terms to differentiate between a standard configuration and an energy model configuration.) An organism's metabolic rate is set so that if the organism executes a user-defined maximum number of instructions, that organism's energy will be depleted and the organism will die. In all energy model experimental runs described in Section 2.4, this number is set to 2000. Therefore, an organism can execute at most 2000 instructions within its lifetime.

$$metabolic\ rate = \frac{stored\ energy}{maximum\ instructions\ executed} \tag{2.1}$$

Instruction Costs. An organism's metabolic rate is used to determine the energy cost associated with executing an instruction. As shown in Equation 2.2, the actual energy cost of executing an instruction is proportional to an organism's metabolic rate. Specifically, the actual energy cost paid by an organism for executing an instruction is calculated by multiplying the organism's metabolic rate by a user defined energy cost for the instruction being executed. Instructions can be assigned different energy costs if desired. An organism with a higher metabolic rate will pay more to execute an instruction than one whose rate is lower.

$$actual\ instruction\ energy\ cost = metabolic\ rate \times instruction\ energy\ cost \tag{2.2}$$

Equation 2.2 produces an evolutionary pressure to lower an organism's metabolic rate so to conserve energy. In contrast, Equation 2.1 causes the evolutionary process to favor organisms with larger quantities of stored energy, because those organisms are scheduled more often. The combination of Equation 2.1 and 2.2 produce a tradeoff where selection attempts to both maximize metabolic rate while minimizing energy cost per executed instruction.

In the following experiment three different configurations are compared to illustrate how the inclusion of energy can affect the evolution of a population. In all configurations the merit or metabolic rate of every organisms is used by the scheduler to probabilistically select the order in which organisms execute.

2.3.5 Real-world and Avida environments

There are many similarities between real-world sensor networks, robotic swarm environments, and the Avida environment. For example, multi-agent systems are common in all three environments. Furthermore, an agent's capabilities are also similar. Agents have the ability to perform local computation and communicate with other agents. In addition, agents can also be mobile. These capabilities can be leveraged and coupled within a group to produce collaborative behaviors enabling the completion of a complex task through the self-organization among individuals.

Energy considerations can play a role in an Avida environment much like they do in a real-world application. In a physical environment, some operations such as movement and communication, are more expensive than others, such as local computation.

With the development of the energy model in Avida and the use of differential instruction costs, different energy expenditures per operation can also be modeled within a digital organism. Doing so allows the evolutionary process to optimize not only behavior, but also total energy usage. For example, a typical wireless sensor network application attempts to extend a node's battery life by limiting its resource usage. The energy model enables the

same goal to be present in Avida. Furthermore, energy can be strictly limited, such that each organism has an energy store that cannot be recharged, similar to the use of a battery in a real-world wireless device.

2.4 Experiments and Results

Resource competition is important to the evolution of a population. In Avida, traditionally, competition for space has driven evolution. The only way an organism is removed from the population is if it dies from excessive age or it is replaced by a newly replicated organism. To ensure the preservation of its genetic material, an organism must replicate before it is replaced. This dynamic is always true even when resources required to complete reactions and gain rewards are limited. However, with the inclusion of energy, an organism can die from starvation or energy depletion. Additionally, a population's size can be constrained by restricting the inflow of resources that are required for reactions. The experiments described in this section demonstrate the different resulting evolutionary outcomes that arise when energy is included in a model. We illustrate differences by evolving populations under various resource availability regimes.

Reaction Types. Avida allows a user to determine the size and type of a bonus received by an organism when it completes a reaction. Traditionally, these bonuses have exponentially increasing values that correspond to the complexity of the task associated with a reaction. The complexity of a task is measured by the minimum number of NANDs required to perform the logic operation that defines the task. In addition, the reward received by the organism is dependent on the availability of a required resource. The completion of a reaction with an underutilized resource will confer a larger reward than the completion of the same reaction when its resource is depleted. In these experiments we compare the effects of changing the type of reward received by an organism.

We compare three different reward regimes: two that use merit-based scheduling where executing an instruction costs only virtual CPU cycles (time), and one that uses metabolic rate-based scheduling where executing an instruction costs both virtual CPU cycles (time) and energy. Using merit-based scheduling, we configure two treatments, which we will refer to as exponential and additive. In the *exponential* treatments, the reward given to an organism for completing a reaction grows exponentially as the requisite tasks become more complex. In the *additive* treatments the rewards for the completion of all reactions are equal, disregarding task complexity, and they accumulate additively. Using metabolic rate-based scheduling, we also configure *energy* treatments, where the rewards for the completion of all reactions are equal, and they accumulate additively as energy. Each treatment consists of an environment containing ten reactions, all of which require a unique task and resource. Table 2.1 shows a list of the tasks and the values of the reward received by the organism for completing the reaction associated with that task. To enable comparisons to previous work, we ran two separate batches using all three scheduling configurations. In one batch we restricted the number of times an organism can complete a reaction to one. In the other, this limitation is removed.

Table 2.1: Tasks required by reactions and treatment dependent bonuses.

Tasks	Exponential	Additive	Energy
ECHO	$2^{0.5}$	10	10
NOT	$2^{1.0}$	10	10
NAND	$2^{1.0}$	10	10
AND	$2^{2.0}$	10	10
ORN	$2^{2.0}$	10	10
OR	$2^{3.0}$	10	10
ANDN	$2^{3.0}$	10	10
NOR	$2^{4.0}$	10	10
XOR	$2^{4.0}$	10	10
EQU	$2^{5.0}$	10	10

Experimental Configuration. The environmental configurations, excluding resource inflow and scheduling method, were held constant in every run. The environmental setup is modeled after a chemostat where both organisms and resources are well mixed. A population exists in a 60×60 torus of cells. Each cell has 8 neighbors. Resources are evenly distributed and every organism is capable of consuming 0.25%, or $9/3600$, of any available resource. This configuration is consistent with previous work [35]. In addition, the resource consumption of an organism is constrained to be between 0.1 and 1 unit to simulate minimum and maximum resource consumption thresholds. Lastly, six resource inflow rates were used in separate treatments. In each of these treatments every resource, excluding the resource required of the ECHO reaction, flowed into the environment at a rate of $0, 1, 10, 100, 1000$ (1K), and $10,000$ (10K) units per update, respectively.

To sustain a population using metabolic rate-based scheduling, organisms in the population must be able to consume a resource at the beginning of a run, otherwise they would die of energy depletion. Therefore, the ancestral organism used to seed a population must be capable of performing a task that can sustain the population. This trait is not required for merit-based scheduling. To control for this difference, in all runs, the ancestral organism is capable of perform the ECHO task. Furthermore, the resource associated with the ECHO task flows into the environment at a constant rate of 100 units per update in all treatments. When using metabolic rate-based scheduling, the combination of the inflowing resource and the organism's endowed ability to perform ECHO sustains a population whose size is limited by the inflow of this resource until organisms evolve to complete other reactions.

The experimental data presented in the remainder of this section are comprised of 18 separate treatments – 6 resource inflow rates for each of the three scheduling methodologies. Each treatment comprises 50 runs, each initiated with a unique random number seed. Each run lasts for $250,000$ updates. An *update* is the measure of time in Avida, during which an average organism will probabilistically execute 30 instructions. Each resource decays at a rate of 1% per update. Additionally, all instructions have been configured with

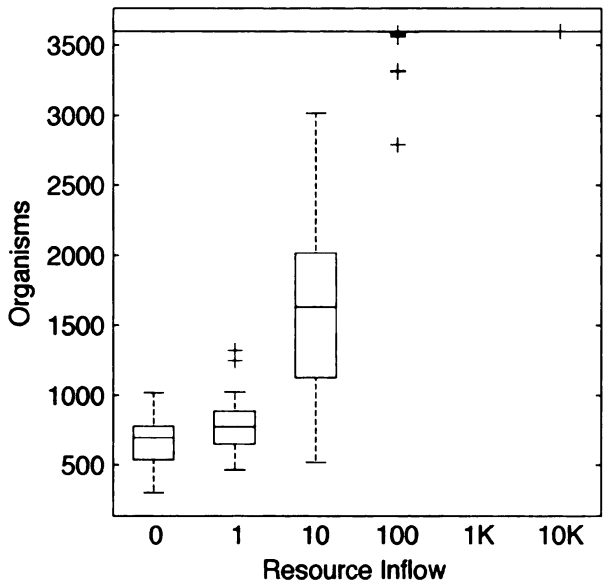
equal time and energy costs.

2.4.1 Population Size Variation

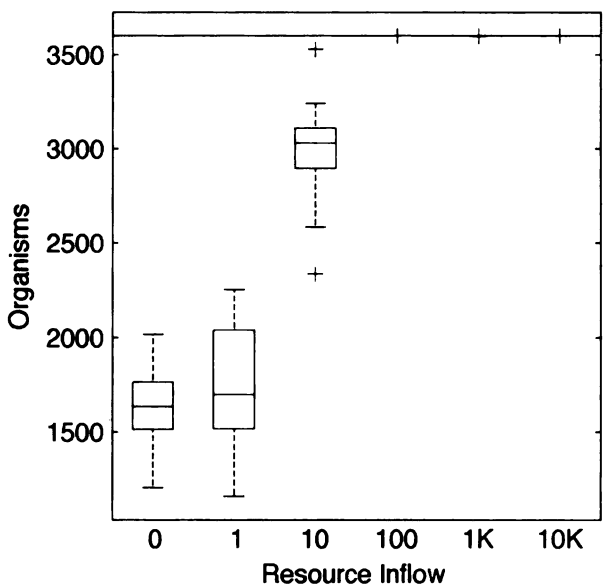
The most obvious difference between treatments that require organisms to pay for instruction execution with time and those that require both time and energy is population size. Configurations that do not require energy all sustain populations at or near the maximum population size, limited only by the available space in the torus. This occurs because, as will be shown in Section 2.4.2, a population's growth rate exceeds its death rate, excluding death caused by one organism replacing another.

By including energy costs, non-space resource limitations can constrain the size of a population. Specifically, the inclusion of an energy cost enables other factors, such as starvation and energy depletion, to increase the population's death rate. Under these conditions an evolving population's size is controlled by the inflow of resources, not the availability of space. Moreover, the size of a population can change as its constituents evolve to collect more energy, more efficiently, by performing more reactions and replicating with fewer instructions.

Focusing on metabolic rate-based scheduling, Figure 2.3 plots the variation in the size of evolved populations at the end of the runs under all inflow rates. Figure 2.3(a) shows the final evolved population size when organisms are constrained to performing each reaction at most once. Figure 2.3(b) shows the same information with the reaction limitations removed. When the resource inflow is zero, the population is living off the resource required to complete the ECHO reaction, as it flows into the environment at a rate of 100 units per update. These populations are the most resource constrained and exhibit the smallest populations. However, the population size is not statistically different from populations evolved with resource inflow rates of 1. A resource inflow of ten also restricts the size of a population in both cases. Higher resource inflows produce populations whose limiting resource is space rather than inflowing resources.



(a) An organism can complete a reaction only once.



(b) No restriction on completion of reactions.

Figure 2.3: Distribution of evolved population size when using metabolic rate-based scheduling.

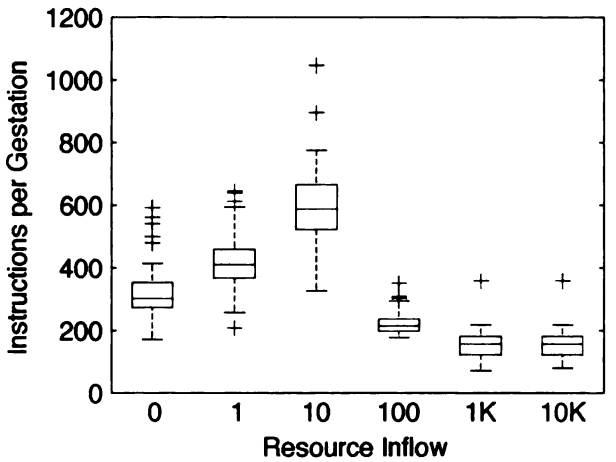
Including an energy cost enables the evolution of populations that can be constrained by resource inflow. This feature may produce dynamics that alter the evolutionary trajectory that a population may take when compared with merit-based scheduling. As demonstrated in Figure 2.3, the size of the population is limited by inflowing resource for inflow rates less than or equal to 10 units per update. Under these conditions none of the runs reached the maximum population size. For the remainder of this chapter, we will focus on three specific aspects of the evolved organisms: their gestation, behavior, and ecological diversity of the population. These aspects allow us to assess competition in each treatment and determine if the organisms evolve to compete for space, inflowing resource, or a combination of both.

2.4.2 Gestation Variation

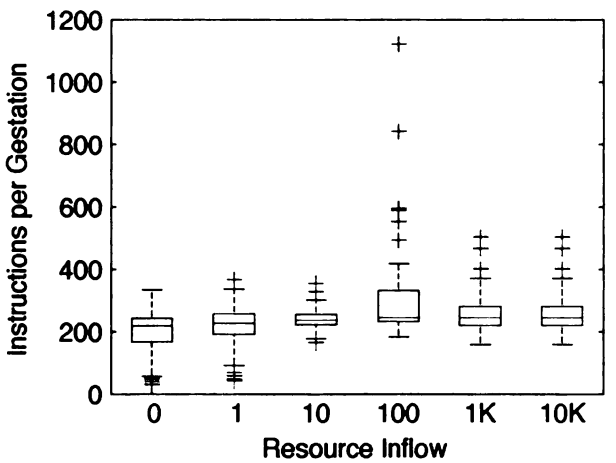
A population's rate of growth may change as its evolves. Limitations on the behavior of organisms can affect this rate. In this section we illustrate the differences in evolved gestation time among the various treatments. We show that an organism's evolved gestation time is dependent on the type of reward it receives when it completes a reaction. We also illustrate that the availability of resources in the energy treatment affects gestation.

Evolutionary pressures can both increase and decrease the gestation of organisms as they evolve. Limitations on rewarding behaviors can have evolutionary consequences. Figure 2.4 shows the evolved average time, measures in virtual CPU cycles, required for an organism to self-replicate, its gestation time, in each treatment when an organism can be rewarded only once for completing a specific reaction. In this figure, the average evolved gestation time in the exponential and additive treatments are relatively constant across all inflow rates. Therefore, varied inflow rates do not affect the evolution of gestation time differently in these treatments. However, this statement does not hold for the energy treatments.

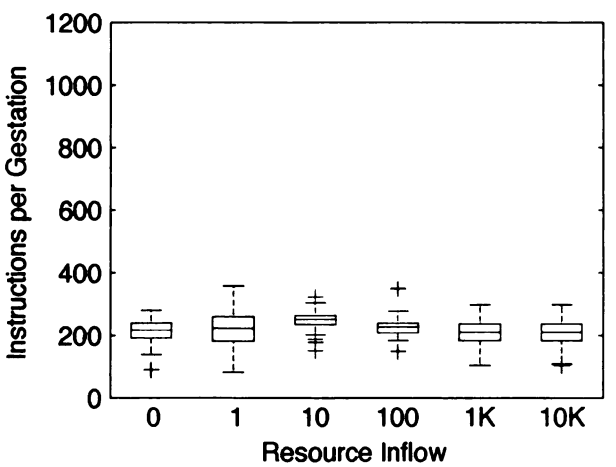
In the energy treatments there is a variation in gestation time that correlates with inflow rate, which correlates with population size. When the evolved population size is below its



(a) Energy



(b) Exponential



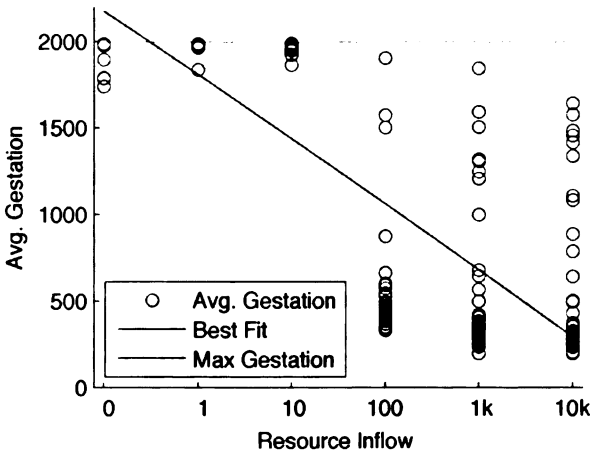
(c) Additive

Figure 2.4: Distribution of evolved gestation time when an organism can complete a reaction only once.

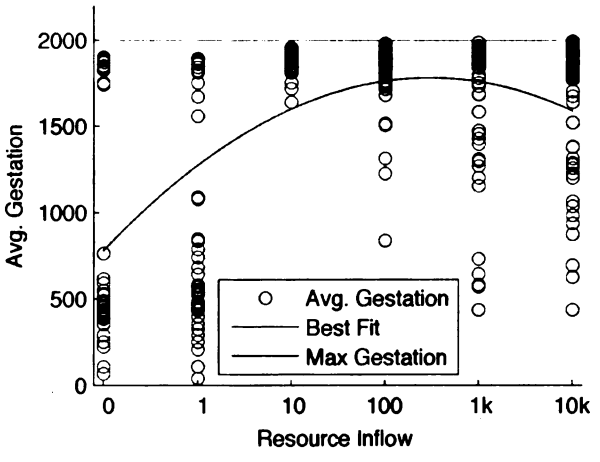
maximum, the evolved gestation time is statistically higher than when space is limiting. For all pair-wise comparisons of resource inflows 0, 1, and 10 to inflows 100, 1000, and 10,000 the maximum p-value, using the Wilcoxon rank sum test for equal medians, was less than 10^{-10}. Therefore, in the three environments with the lowest inflow rates, selection favors longer gestations, which provide organisms with more time to compete for inflowing resources. However, when the restriction on the number of reactions that can be performed is removed, more variation in evolved gestation time is observed. Figure 2.5 shows the evolved average gestation time in each treatment when an organism is allowed to perform a reaction an unlimited number of times. The dashed horizontal line at the top of Figure 2.5 represents the configured maximum allowable age of 2000, which is approached in many runs.

Again, as in Figure 2.4(c), in the additive treatments across all inflow rates we see gestation times that are similar. However, testing all pairwise additive treatments for statistical differences we observe that the gestation times for inflow rates 0 and 1 are significantly different than the other four inflow rates, with a maximum p-value less than 10^{-4}. This statistical difference, albeit small, suggests that the difference in an inflow of 1 and an inflow of 10 is enough to cause an evolutionary pressure that selects for a slight, yet significant, decrease in gestation time.

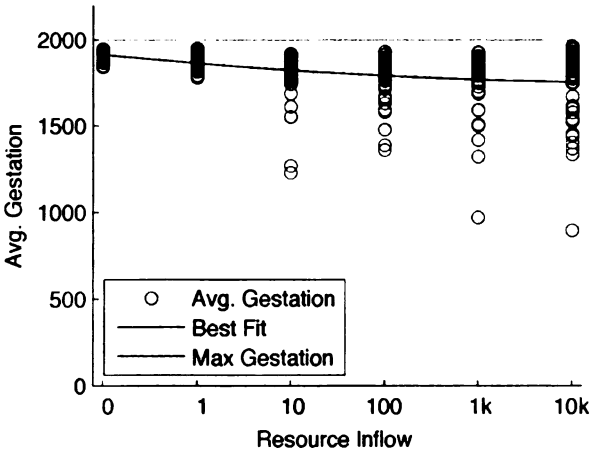
Figure 2.5(a) demonstrates a correlation between gestation and population size in the energy treatments when reactions are not limited. As in Figure 2.4(a), we see gestation times decrease in the energy treatments when space is the limiting resource, *i.e.* resource inflow is 100 or greater. However, when there is ample room for new organisms the average gestation of an organism approaches the maximum allowed. These organisms have evolved to live as long as possible, which provides them with more time to perform tasks that, when resources are available, confer a reward. In addition, the organisms in these populations are not subjected to death by replacement, since there is ample room in the environment, so an extended gestation is a viable evolutionary solution.



(a) Energy



(b) Exponential



(c) Additive

Figure 2.5: Average organisms gestation time in each run with no reaction restrictions. Dashed horizontal line at 2000 represents the maximum possible gestation.

The data in each plot in Figure 2.5 are fitted to a third-order polynomial. In all three plots the trends of these lines differ. In the additive treatment, shown in Figure 2.5(c), the line-of-best-fit illustrates that resource inflow has little effect on evolved gestation times. However, in the energy treatment, shown in Figure 2.5(a), as resource inflow increases, the average gestation of an organism declines from approximately 2000 instructions to 300 instructions, almost an order of magnitude decline. Moreover, a large decrease occurs between inflow rates of 10 and 100, which coincides with the tipping point where a population is restricted by inflowing resources and those that are limited by space. Lastly, the evolved gestation times on the exponential treatments are bifurcated under 0 and 1 resource inflow rates, clustered at an inflow rate of 10, and skewed at higher inflow rates. Because of the variation in evolved gestation times under the various inflow rates in the exponential treatments, we are unable to precisely state how resource inflow affects the evolution of gestation times. However, in the following subsections we do discuss the bifurcation observed under low resource inflows.

Gestation Bifurcation

Figure 2.5(b) depicts a wide variation in gestation times within and across the exponential treatments. Focusing on the inflow rate of 0, we observe populations that have evolved to exhibit average gestation times as high as 1902.0 and as low as 65.7. Furthermore, no populations evolved average gestation times in between 764.6 and 1742.6. This bifurcation of evolved gestation times suggests that selection does not strongly favor longer or shorter gestations. To augment our view of gestation in this treatment we plot the average gestation time of each run over the course of evolution in Figure 2.6. Figure 2.6 clearly shows a bifurcation of average gestation time, which occurs early for the majority of runs. The scatter plot on the right side of Figure 2.6 depicts the final average gestation time for each run, duplicating data shown in Figure 2.5(b), as well as a count of the points in each of the two clusters. Out of 50 runs, 64% end with a gestation time in the lower cluster, between

65.7 and 764.6 instructions. The remainder have a higher gestation time, between 1742.6 and 1902.0 instructions. Additionally, Figure 2.6 shows that a switch from a higher average gestation to one that is lower is possible and most likely to occur early in the evolutionary process. After 150,000 updates have passed there are no transitions from a high gestation to a low gestation. These data show, generally, runs that evolve a higher average gestation remain high, however there is a potential for a fortuitous mutation(s) to cause the population to switch to a lower average gestation. A similar bifurcation is also present in the data for an inflow of 1, shown in Figure 2.5. The data for the other four inflow rates do not exhibit a bifurcation. They are clustered around a high gestation time, and the three highest inflow rates have a long skew in the lower gestation direction.

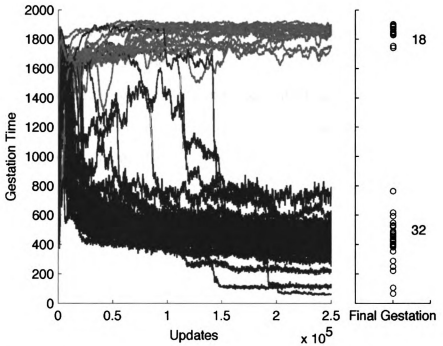


Figure 2.6: Average gestation time over evolutionary time of all 50 runs in the exponential treatment with an inflow rate of 0 and no reaction on restriction completion (left). Black lines represent runs whose average gestation time was low at the end of the run. Lighter colored lines are runs whose gestation was high at the end of the run. Final average gestation of each run with count of runs in each cluster is shown in the scatter plot on the right.

Behavior Bifurcation

The existence of the gestation bifurcation, show in Figure 2.6, results from conflicting selective pressures for low and high gestations. The gestational history of the populations can be categorized into three groups: remains low, remains high, or transitions from high to low. Out of the 50 runs, none transitioned from low to high, however one run does come close.

Before discussing the evolutionary pressures that produce the three categories of gestational history, we must first emphasize the environmental conditions under which these populations evolved. These populations are from the exponential treatment when only the resource required for the ECHO reaction flows into the environment. Therefore, no other reaction is rewarded. Under these experimental conditions a population is at or near full and the resource required to complete the ECHO reaction is depleted, so some completions of the ECHO task are not rewarded.

The conditions in this environment produce conflicting evolutionary pressures. First, the competition for space, which is present because an organism can replicate without completing any reaction and the torus has a fixed size, produces a pressure to replicate quickly. This pressure can manifest in organisms with shorter gestation times or organisms that perform more reactions, which results in preferential scheduling. Second, the competition for resources produces a pressure to perform more reactions, consuming more resources and gaining merit. Combined, these two pressures could result in organisms that evolve to attempt many reactions within a short gestation time, thereby predicting the existence of the lower gestation time cluster. To explore this hypothesis we compare the average evolved gestation times and fitness of organisms in the two clusters. The average gestation time of runs within the lower cluster is 426.9 instructions, which is 4.3 times lower than the upper cluster's average gestation time of 1847.6. A lower gestation time is a distinct advantage in a space-constrained environment because an organism can replicate more quickly, spreading its genetic material through the population before it is replaced.

In fact, the runs in the lower cluster have a significantly higher fitness when compared to the runs in the higher cluster (p-value less than 10^{-8}). Moreover, the average fitness of organisms with lower gestation times is approximately 42% higher than those with higher evolved gestation times.

However, if true, the speculation above explains only half of Figure 2.6. It does not explain why 36% (18 out of 50) of the runs evolved longer gestation times. One potential explanation is that the scheduling methodology influenced the results. Specifically, selection may favor organisms that evolve a longer gestation time because they have more time to consume the resource required to perform the ECHO reaction, increasing their merit more per generation. Using merit-based scheduling, this would result in organisms that are scheduled more often. Is this a strong enough selective pressure to cause the bifurcation observed in Figure 2.6? We argue that it is. Calculating the median number of reactions performed by an organism at the end of a run, and dividing by the number of instructions executed, we observe that an average organism with a gestation time in the higher cluster perform 0.0102 reactions per instruction, while those with lower gestation times perform 0.0100 reactions per instruction. This difference, though small, is significant, with a p-value of 4.8948×10^{-4} using the Wilcoxon rank sum test for equal medians. In addition, the number of reactions performed by an average organism in the upper cluster is 4.4 times larger than that of an average organisms in the lower cluster.

The dichotomy present in the evolved gestation time in the exponential treatment is not present in either the additive or energy treatments. In the energy treatment there is no competition for space since the population is not full for inflow rates of 10 or below. Therefore, newly born organisms can alway be placed into an empty cell and no existing organisms is ever replaced. This lack of space competition forces selection to act on the collection and use of resources, which leads to long gestation times, as shown in Figure 2.5(a). The additive treatment also evolves organisms with long gestation times. In these runs there is competition for space, however selections favors organisms that live longer and

have the opportunity to complete more reactions than organisms with shorter gestations.

2.4.3 Behavior

As the populations in our treatments evolve, two different aspects of an organism are easily observable and directly relate to the comparison of different scheduling methodologies. The first is the gestation time of an organism, discussed above. The second is the set of reactions that organisms perform. This section examines the effects that the three scheduling methodologies and varied resource inflow rates have on the evolution of the number of reactions performed. It also illustrates the effects the treatments have on ecotype diversity. We show that all treatments exhibit increases in reaction completion across resource inflows.

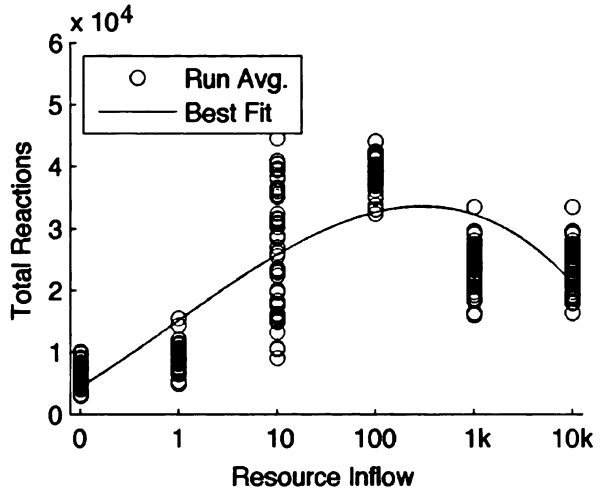
Resource inflow does have an effect on the number of reactions performed by organisms within a population. This effect is due in large part to the prerequisite resource requirements that have been placed on reactions, which simulate physical restrictions on the reactions. For a reaction to be triggered, a minimum amount of the required resource must be available. In addition, the amount of resource consumed by completing a reaction is also limited, so that a single organisms can only consume a maximum amount per reaction. The restriction on the number of times an organism may complete a reaction also limits the number of reactions performed.

Figure 2.7 plots the number of reactions performed at the end of all runs in all three treatments where an organism can complete a reaction only once. These data are fitted to a 3rd-order polynomial. In Figure 2.7, inflow rates of 100 or below display increases in reaction completion as resource inflow increases. This increase is expected because as resource inflows increase the minimum resource restriction becomes less of a factor in reaction completion. However, this trend does not persist across all inflow rates. Specifically, there is a statistically significant decline in total number of reactions completed between resource inflows of 100 and 1K in all treatments, with p-values (calculated using the Wilcoxon

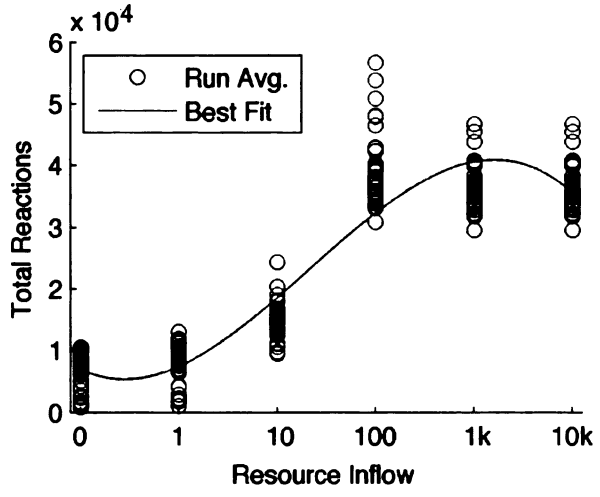
rank sum test for equal medians) of 5.3181×10^{-17}, 0.0253, and 8.9526×10^{-11} for the energy, exponential, and additive treatments, respectively. This decrease in reaction completion is a result of the restriction on the number of times a reaction can be completed, and is not present when this restriction is removed.

Focusing on the energy treatments, Figure 2.8 shows the distribution of the sum of all available resources for all inflow rates. For inflow rates of 100 or less, most of the populations have available resource levels within the maximum and minimum consumption range, depicted in Figure 2.8. These lines depict the region of resource space where a population could consume all of the resources in its environment if every organism completed a reaction that consumed the maximum or minimum amount of resource. These resource limited treatments correspond to the treatments where reaction completion increased as resource inflow increased, as shown in Figure 2.7(a). Therefore, the amount of resource in the environment is limiting the number of reaction performed in these treatments. However, when the inflow rate is 1K or greater, resources are not limiting the number of reactions performed by the organisms. Instead, the organisms are limited only by the reaction restriction, which was also present in the treatments with lower resource level. So, we might ask why the number of reactions declines at higher inflow rates. At inflow rates of 1K or greater the average gestation of an organism is significantly less than for all lower inflow rates, with a maximum p-value less than 10^{-11} when comparing inflow rates of 100 and 1K using the Wilcoxon rank sum test. Therefore, at these high inflow rates the competition for space is stronger and selection favors replicating more quickly over completing more tasks.

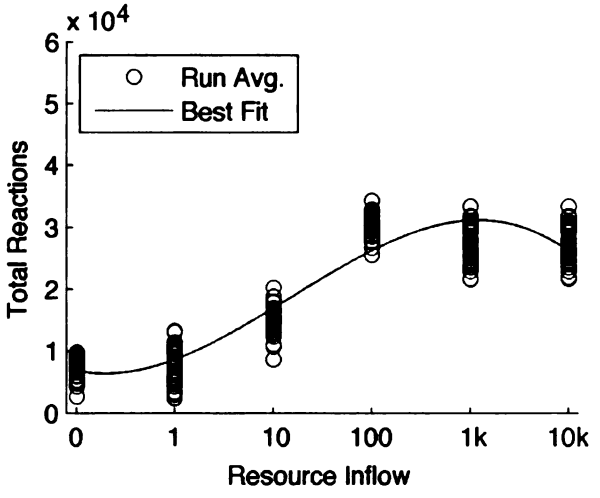
When the limitation on performing a reaction only once is removed, an evolved population performs increasingly more reactions with diminishing gains as resource inflows increase, as shown in Figure 2.9. This trend mirrors that observed on the left side of Figure 2.7. However, there are no declines in reaction completion, which are present when the reaction restriction is present. These data suggest that even at a high resource inflow, when



(a) Energy



(b) Exponential



(c) Additive

Figure 2.7: Total reactions performed when each reaction can be performed only once per organism.

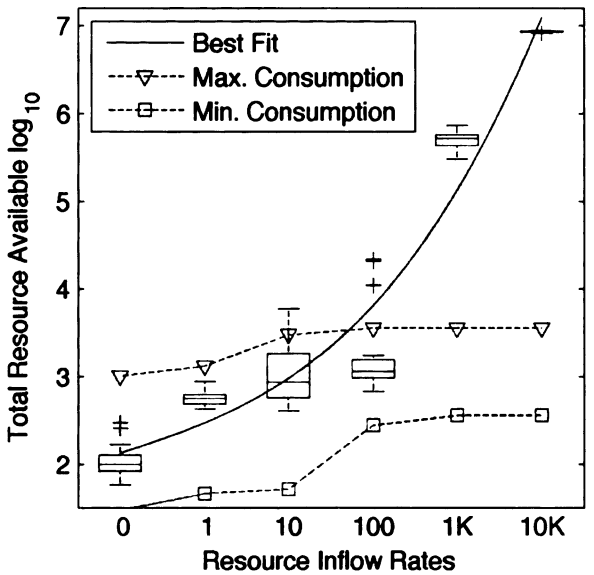
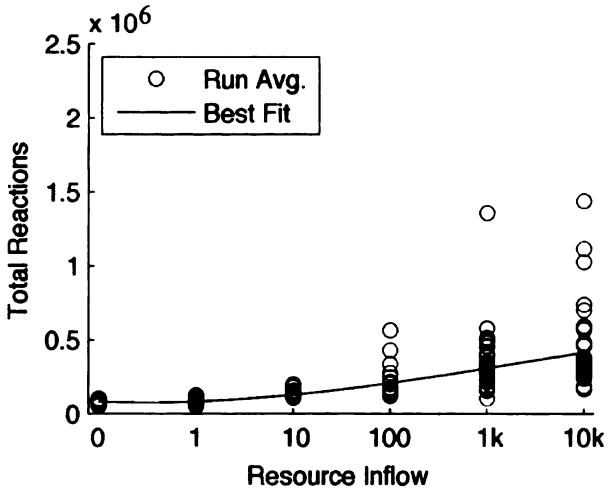


Figure 2.8: Variation of resources that are required for reactions when each reaction can be completed only once per organism.

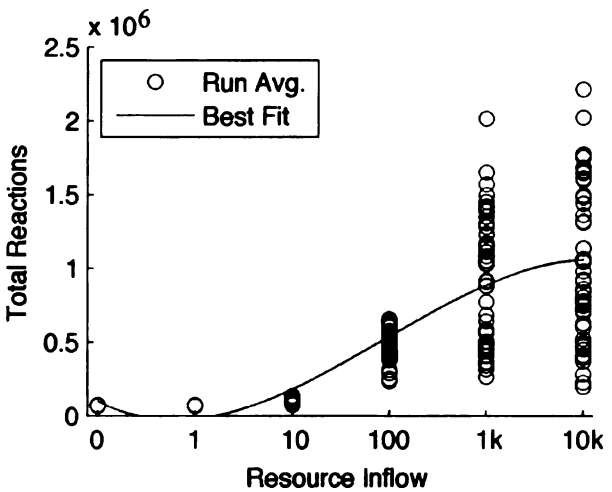
reaction completion is not restricted (see Figure 2.10), there remains a selective pressure to complete a large number of reactions.

Ecotype Diversity

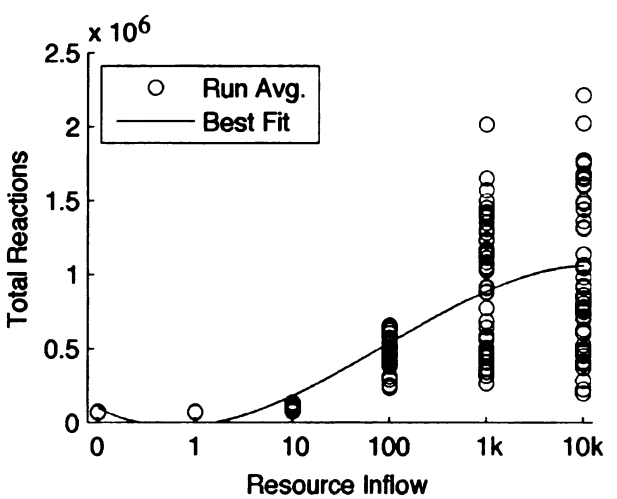
This section illustrates how feedback affects ecological diversity within populations in our treatments. We measure ecological diversity by grouping genomes based on the reactions an organism with that genome will perform when executed and then count these groups. Each organisms in a group performs the same set of reactions, its *phenotype*. For example, one group may contain genomes that perform only ECHO, while another group may perform ECHO and NAND. We refer to each of these groups as an *ecotype*. To be counted as an ecotype, a minimum of ten genomes must perform the same set of reactions, regardless of quantity, and produce viable offspring. We ignore quantity of reactions performed so that a comparison between treatments with reaction restrictions and those without can be made. A viable organism is an organism that produces an organism that can reproduce. In



(a) Energy



(b) Exponential



(c) Additive

Figure 2.9: Total reactions performed when reaction completion restrictions are removed.

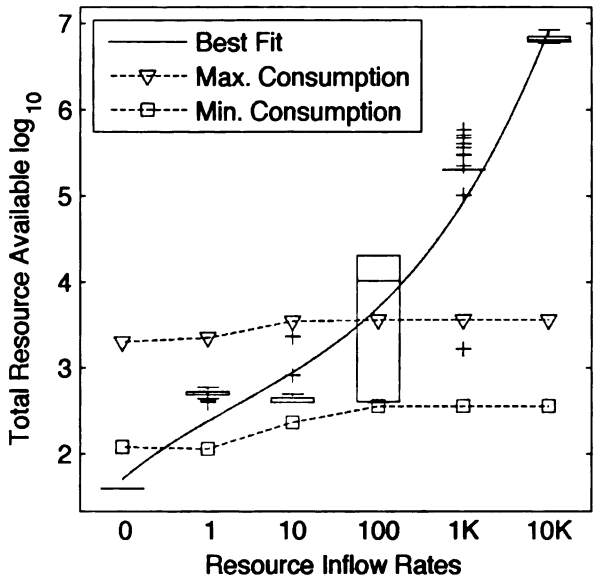


Figure 2.10: Variation of resources that are required for specific reactions when reaction limitation is not present.

total, there are 2^{10} possible ecotypes that could evolve in every treatment.

The performance of a reaction is dependent on the availability of a resource. The availability of a resource depends on past consumption and inflow rate. If no reactions are performed and all resources flow in and decay at the same rate, producing equivalent resource levels, then selection would not favor the consumption of one resource over any others. The top of Figure 2.11 illustrates this concept by showing the transition from using one resource to another is equally likely. However, once the completion of a task is associated with the consumption of a resource, as it is in these experiments, then selection can act on organisms that consume resources. For example, as demonstrated in the bottom of Figure 2.11, selection favors organisms that perform reactions that consume underutilized resources, such as Resource 0, over those that consume depleted resources, such as Resource 1, because the reward for completing the reaction that consumes Resource 1 declines along with the quantity of the resource. Therefore, the overutilization of a resource will negatively affect the consumers of that resource and promote the utilization of other

underutilized resources.

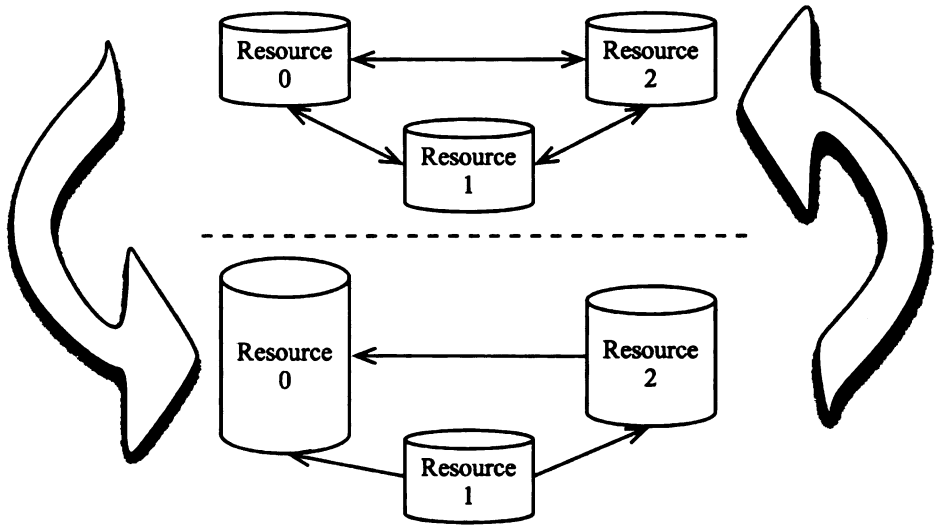


Figure 2.11: Resource equalization cycle with lines that point to resources whose utilization is selected for.

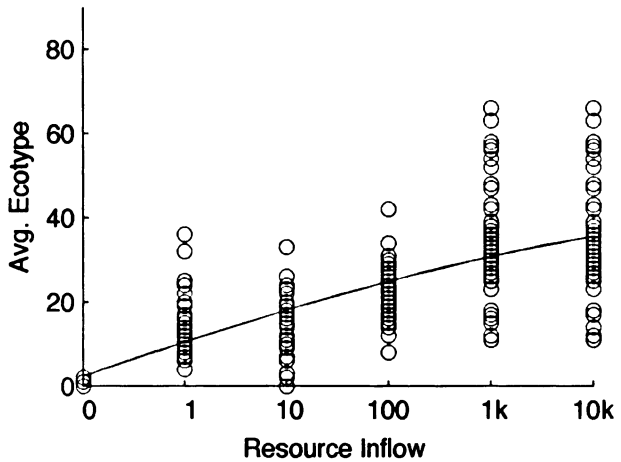
The negative feedback loop in Figure 2.11 is generated by the overutilization of some resources and the underutilization of others. Uneven resource consumption promotes the selection of individuals that use other resources and stabilizes resource levels as long as consuming a resource is equally as difficult as consuming any other resource. However, in these experiments the complexity of tasks required to consume a resource varies. Some resources will be consumed more often than others because it is easier for the evolutionary process to evolve organisms that do so. As the evolutionary process produces organisms that consume these "easily" consumable resources, those resources will be depleted and selection will drive the population toward the utilization of other resources associated with more complex tasks. Over evolutionary time more resources will become utilized by organisms that perform a range of reactions.

Figure 2.12 and 2.13 display scatter plots of the ecotype diversity at the end of every run fitted to a 3rd order polynomial for treatments without reaction restrictions and with reaction restrictions, respectively. In Figures 2.12(b) and 2.12(c) the number of ecotypes observed is highest at an intermediate resource inflow. This result is supported by previous

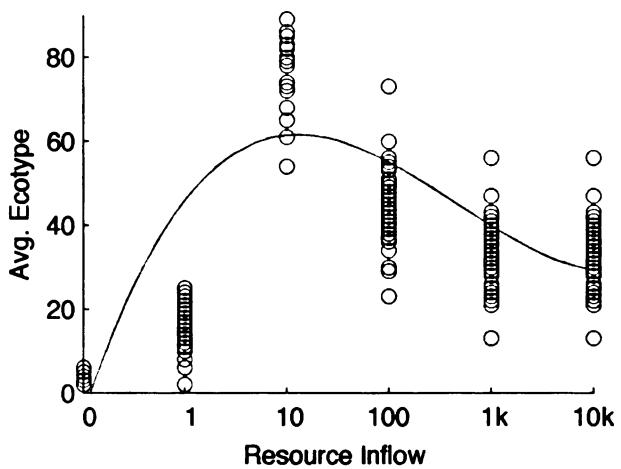
work [35]. However, this trend is not present in the energy data displayed in Figure 2.12(a). These data illustrate a consistent, yet diminishing, increase in ecotype diversity as inflow rate increases. This increase is a result of the restriction on the performance of reactions, and is not present when the restriction is lifted, as in Figure 2.13(a). The trend of increased ecotype diversity at an intermediate inflow rate is apparent in all three subfigures. However, the average ecotype diversity is lower when the reaction limitation is not enforced. This is a result of the selective pressure to perform more reactions produced when the number of reactions an organisms performs is limited.

2.5 Discussion and Conclusions

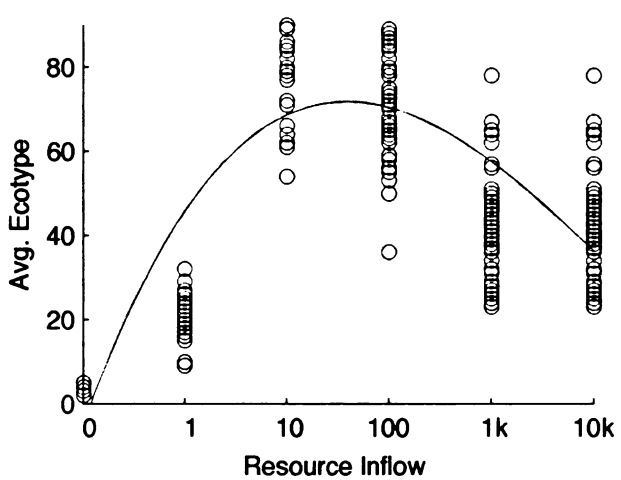
This section illustrates similarities and difference that result as a byproduct of the type of reaction rewards that are employed by the experimenter. We have shown that the incorporation of the energy model produces results that are consistent with existing evolutionary theory and empirical data. We have shown that the energy model can be used to control a population's size, and we have demonstrated that energy costs do not adversely affect ecotype diversity. Additionally, we demonstrated, in low resource inflow environments, that an exponential reward structure can produce divergent results. Building on these baseline experiments, we will now demonstrate the application of the energy model to the evolution of individual and group behaviors.



(a) Energy

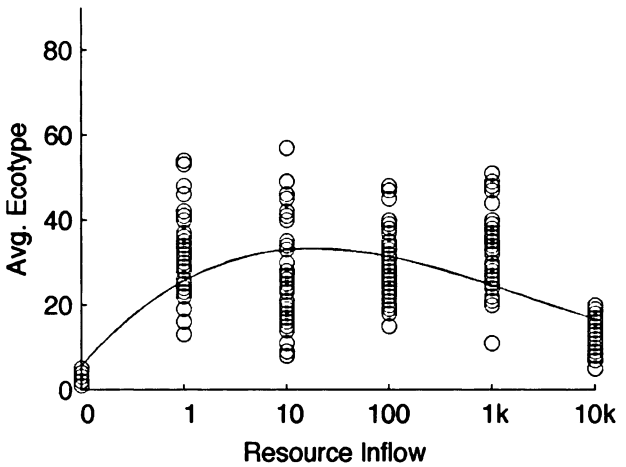


(b) Exponential

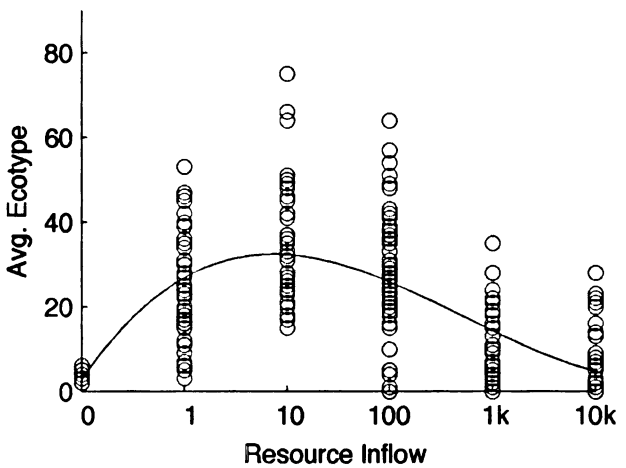


(c) Additive

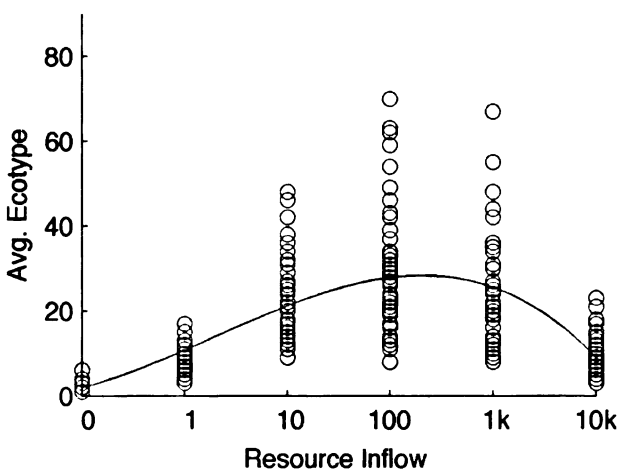
Figure 2.12: Ecotype diversity of all runs when an organism can perform a reaction only once.



(a) Energy



(b) Exponential



(c) Additive

Figure 2.13: Ecotype diversity of all runs without reaction restrictions.

Chapter 3

Individual Energy Management

In this chapter we investigate the evolution of individual energy management. Having shown the affects of the inclusion of energy costs, we now focus on the evolution of resource aware behavior. Specifically, we will show that digital organisms are capable of evolving adaptive sleep/awake behavior and phototaxis.

3.1 Evolving Organisms that Sleep

A population of organisms in an environment where a resource is always available can be non-adaptive and function exceptionally well. There is little or no selective pressure on the organisms to adjust their behavior within this environment since resources are plentiful and can be consumed at any time [101]. If resources can become diminished or unavailable, however, an adaptive response might allow for more conservative resource usage [184] or increased energy storage [92]. Natural organisms often display adaptive behavior that coincides with environmental changes where resources fluctuate [50, 97]. An example of this type of adaptive response occurs in nocturnal rodents and insects, which sleep during the day and forage for food under the cover of darkness. Animals that hibernate also display an adaptability that allows them to survive extended periods of low resource availability by increasing the size of their fat stores prior to hibernation [198].

This form of adaptive behavior in natural organisms serves multiple purposes. During sleep periods an animal rests [199], reprograms its brain [38] and performs internal maintenance tasks [136]. However, while an animal is in a state of slumber it is less aware of its environment. How could resource-aware adaptive behaviors, such as sleep and hibernation, have evolved in competitive environments where torpid organisms are vulnerable to active organisms? Is there a selective pressure to sleep caused by resource limitations in environments with periodic resource availability? The remainder of this chapter attempts to answer these questions through experiments with digital organisms.

Previous work has been done in this area using neural networks [133]. In [133], the organisms were subjected to two different environments with periodic light availability, where the organism's ability to find a resource was impaired relative to the current light intensity. It was shown that in an environment where light readings may not correctly disambiguate day from night the combination of a biological clock and light sensor produced individuals that could disambiguate night from day. Our study differs from [133] in that it does not impose a predefined structure on the organisms, provide a common starting point to the organisms, or give any information, ambiguous or not, to the organisms directly. All of these mechanisms must be evolved while preserving an organism's ability to self-replicate and while avoiding other detrimental behavioral changes. We begin with a brief overview of extensions to Avida and the experimental setup, followed by presentation of the experimental results.

3.1.1 Experimental Setup

In these experiments, the population of digital organisms is arranged in a 60×60 grid. When an instruction is copied there is a 0.75% chance that it will be mutated. During replication there is a 5% chance an instruction will be deleted from the genome, and a 5% chance that a random instruction will be inserted. On average each organism in the population will execute one instruction per *update*, the standard unit of time in Avida.

As in [116], organisms are rewarded for performing tasks that are Boolean logic operations. Specifically, we used the five tasks listed in Table 3.1. Each task has an associated reward, indicating the number of energy units an organism gains when completed, and a limit on how many times an individual organism can be rewarded for performing it. Completing even these relatively simple tasks can require several instructions. Table 3.2 shows a "hand-built" solution for the AND task (a NOP instruction modifies the behavior of the preceding instruction, for example, placing the result in a different register than the default). Of course, evolution may produce many different solutions for the same task.

Table 3.1: Rewarded tasks.

Task Name	Input	Bitwise Output	Reward	Max Times Rewarded
ECHO	A	A	1000	35
NAND	A,B	$\neg(A \wedge B)$	1500	20
NOT	A	$\neg A$	1500	20
ORNOT	A,B	$A \vee (\neg B)$	2000	13
AND	A,B	$A \wedge B$	2000	13

Table 3.2: Instruction sequence that when executed completes the AND task.

Instruction	AX	BX	CX	Stacks 1,2	Output	Description
IO	?	X	?	?,?	?	read X into BX
IO nop-C	?	X	Y	?,?	?	read Y into CX
nand	?	X nand Y	Y	?,?	–	BX ← ¬(AX ∧ BX)
push	?	X nand Y	Y	X nand Y, ?	–	push BX on stack 1
pop nop-C	?	X nand Y	X nand Y	?,?	–	pop stack, place result in CX
nand	?	X and Y	X nand Y	?,?	–	BX ← ¬(BX ∧ CX)
IO	?	Z	X nand Y	?,?	X and Y	output BX

The environment contains a single resource that is available periodically. When the resource is available, it is non-depletable, and all five tasks described in Table 3.1 are maximally rewarded. If an organism completes a task when the resource is unavailable, no reward is given. The duration of the resource availability changes throughout every experiment except the control experiment, where it remains constant. Resource availability

is defined in "years" and "days." Each year consists of 500 days, each of which lasts for 256 updates. During each year, the availability of the resource remains constant. That is, each day of a year has the same duration of resource availability. At the beginning of each day the resource becomes available for a period of time depending on the current year. For the first year the resource is available during 100% of the day. After each passing year, the availability of the resource during a day is reduced by 6.25% of a full day's length until it becomes zero, which deprives the population of energy and eventually brings on its demise. Through evolutionary change brought upon by depriving the population in this manner, we are able to observe under which conditions the population of digital organisms will find sleep useful.

We have added six instructions to the base Avida instruction set, enabling an organism to sense and respond to its environment. These instructions are: TIME, SENSE, and four variations of SLEEP. Executing the TIME instruction stores the current time step in a register within the organism's virtual CPU. The SENSE instruction allows an organism to detect the presence or absence of the resource; it loads one of the calling organism's registers with the current quantity of the resource times 100. (The value of the resource is multiplied by 100 to allow for a wider range of the sensed value.) The SLEEP instructions allow organisms to enter a low energy state that lasts for multiple CPU cycles. Compared to other instructions, the SLEEP1-4 instructions cost 100 times less energy to execute and last for 10, 20, 40, and 80 times more CPU cycles, respectively.

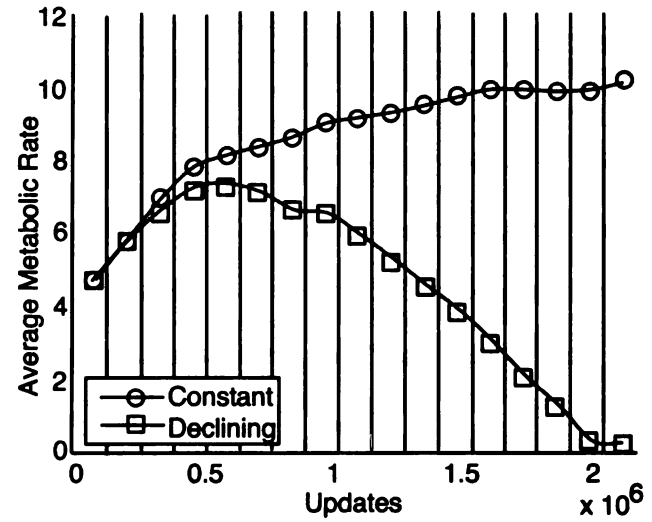
3.1.2 Experimental Results and Discussion

We define an environment where a resource is available for all or part of each day. If the resource is always available, the organisms in the population do not benefit from an adaptive response based on the availability of the resource because the resource can be used at any time. We hypothesize that a decline in resource availability within a single-resource environment can produce an adaptive, resource-aware response.

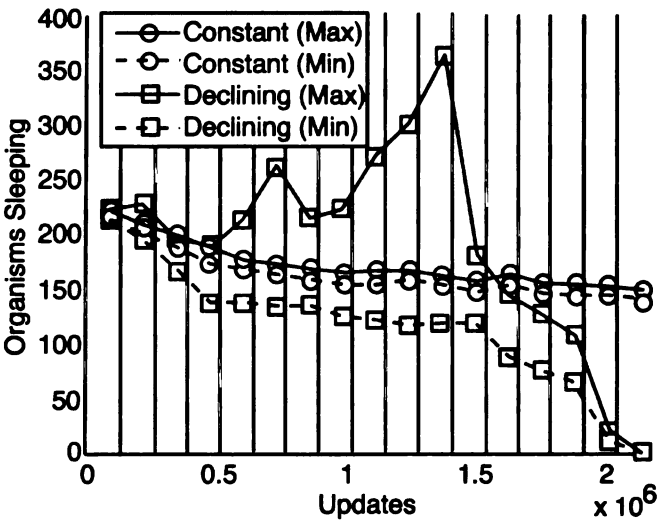
To test this hypothesis we conducted two experiments; results presented are the average of 50 runs. In the control experiment the resource is available during the entire run (constant environment), and in a second experiment the availability of the resource is reduced over the course of the run (declining environment). Figure 3.1(a) displays the average metabolic rate in both the constant and declining resource environments. For clarity, error bars are omitted; the maximum standard error is 0.018 for constant environment and 0.01 for declining environment. The 16 vertical lines in Figure 3.1(a) denote years, where a 6.25% decrease in resource availability occurs in the declining resource environment. As shown, the metabolic rate in the constant environment tends to stabilize as the run proceeds, but decreases over time in the environment with declining resource availability. This behavior is expected, since organisms can receive rewards for completing tasks continually in the constant environment, but less often as time lapses in the declining resource environment. In fact, after the last vertical line the organisms in the declining resource environment populations no longer have a source of energy, and eventually the population will die off.

Figure 3.1(b) shows the average maximum and minimum number of organisms sleeping at some time during a day in each environment. In the constant environment these numbers remain relatively close together throughout the run. In contrast, organisms in the declining resource environment have evolved to produce periods of relative inactivity, where at the peak, on average, greater than 10% of the organisms in the population are sleeping. At this point the number of organisms sleeping in the declining environment is significantly greater than the number sleeping in the constant experiment (p-value < 0.0003, using Wilcoxon rank sum test for equal medians). A sample of evolved code from one of the runs is given in Table 3.3. The code produces a resource-aware behavior when executed. Specifically, the organism enters a loop that ends when the resource becomes available.

Since the organisms sleep more in the declining resource environment, one might infer that the organisms accumulate more SLEEP instructions in their genomes. However, this proved not to be the case. Figure 3.2(a) shows the number of SLEEP instructions that are



(a) Average metabolic rate



(b) Average maximum/minimum sleeping organisms

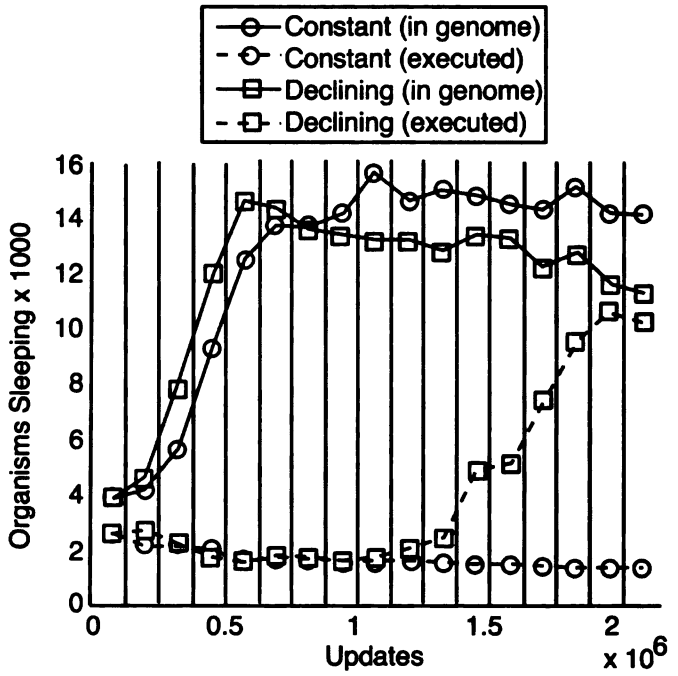
Figure 3.1: Comparison of sleep responses in two environments, one where the resource is available 100% of the time (constant), and one where the resource availability decreases over time (declining). Results are the average of 50 runs.

Table 3.3: Evolved code that loops until the resource becomes available.

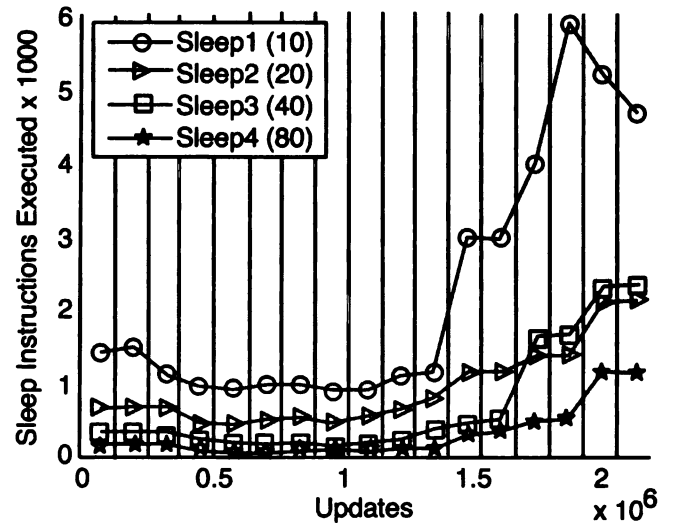
Instruction	Explanation
H-SEARCH	place FLOW-HEAD at next instruction
SLEEP	start sleeping
SENSE	read resource availability into BX register
IF-EQU-0	if BX $\neq 0$ skip next instruction
MOV-HEAD	move INSTRUCTION-HEAD to FLOW-HEAD

present in the organisms' genomes in both environments, along with the number of sleep instructions *executed* in each. For the first half of the runs, organisms in both environments have substantially more SLEEP instructions in their genomes than they actually execute. The gap then begins to narrow in the declining environment, and by the end of the runs the number of executions nearly equals the number present. The increase in the execution of SLEEP instructions in this environment suggests that sleeping is increasingly beneficial as the resource availability diminishes. Figure 3.2(b) shows the rate of execution of SLEEP instructions over the course of the runs in the declining resource environment. As expected, the SLEEP instructions with lower CPU cycle costs are used more heavily than the more expensive SLEEP instructions, especially early in the runs. As the resource becomes scarce, the number of more expensive SLEEP instructions increases. This adaptation allows for longer sleep cycles and fewer executed instructions.

When Avida organisms are exposed to an environment where resource availability varies during a day, they evolve an adaptive resource-aware response. An example is shown in Figure 3.3, which depicts snapshots of the 60×60 grid during a single day in a population that evolved this adaptive sleep/wake behavior. The black squares represent organisms that are sleeping. At this point in the run, the resource is available for the first 112 (out of 256) updates of a day. Figure 3.3(a) shows the population at the beginning of a day. Figure 3.3(d) shows the population at the day's midway point where the resource is no longer available and organisms are beginning to enter a sleep cycle. During this day the peak number of organisms sleeping at one time is 2111 or 58.6%, shown in Figure 3.3(e). After this point the organisms start to wake up and await the next period of resource availability.



(a) number of SLEEP instructions present in and executed by organisms



(b) number of four SLEEP instructions executed, declining environment.

Figure 3.2: Number of SLEEP instructions present in and executed by organisms in the constant and declining environments. Average over 50 runs.

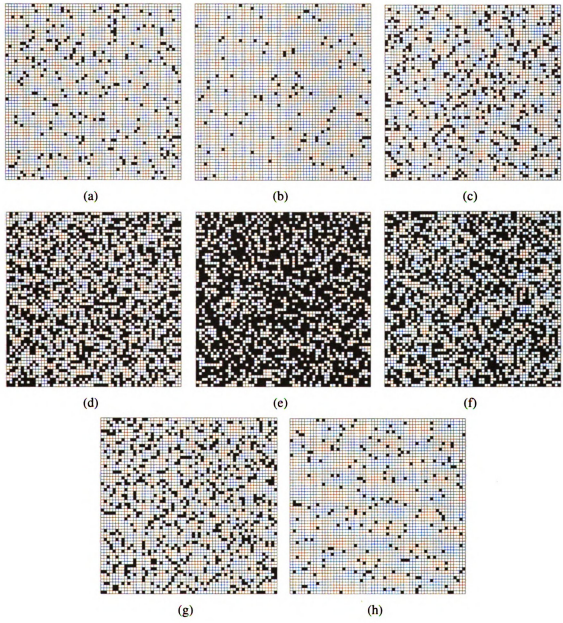


Figure 3.3: Representations of a population's response to the resource availability over a single 256 time-step day. Black squares represent sleeping organisms and white squares represent awake organisms. The resource is available for the first 112 time steps. a) t = 1, 231 sleeping, resource becomes available; b) t = 64, 108 sleeping; c) t = 128, 469 sleeping; d) t = 152, 1355 sleeping, resource is no longer available; e) t = 180, 2111 sleeping; f) t = 204, 1502 sleeping, organisms are beginning to wake up; g) t = 228, 667 sleeping; h) t = 256, 189 sleeping, day ends and resource becomes available again.

Figure 3.4(a) plots the number of organisms sleeping and the resource availability during three consecutive days near the midpoint of a single run, when the resource is available during the first 50% of each day. As shown, there is a tight correlation between number of sleeping organisms and lack of resources. This behavior is reminiscent of circadian rhythms exhibited by many species in the natural world.

Moreover, examination of evolved genomes shows that organisms in this population have evolved to begin their sleep cycle just before the beginning of resource deprived periods, and begin preparing data to be used in tasks, just prior to the return of the resource. This "early to bed, early to rise" behavior allows organisms to finish tasks early during periods of resource availability, thereby increasing the probability of receiving a reward. It also helps to avoid situations where an organism's execution is delayed, causing a task to be completed just after the resource disappears, in which case the organism receives no reward. This adaptive behavior arose in 37 out of 50 runs in the declining resource environment.

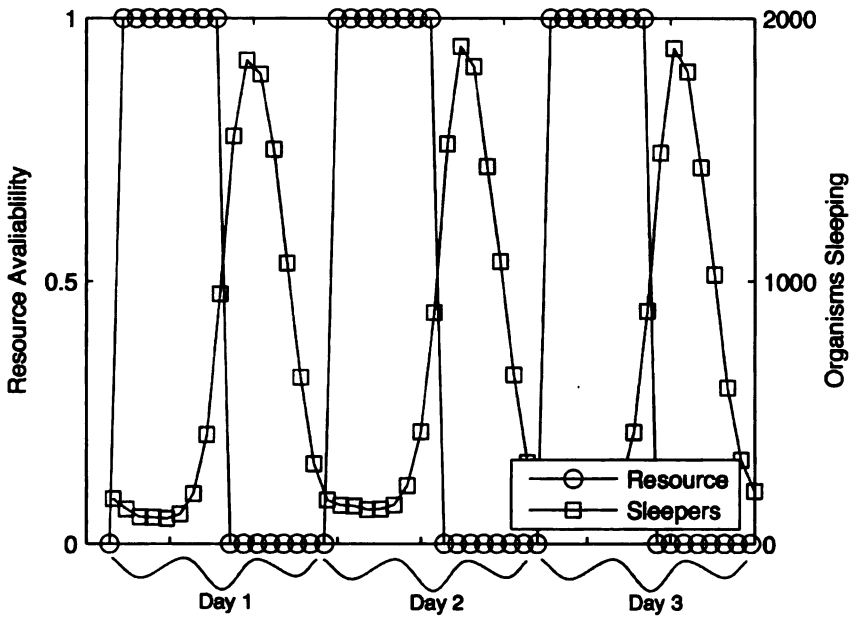
Although the populations evolved an adaptive behavior, in the above trials the fraction of concurrent sleeping organisms never stabilized above 60%. To help explain why more organisms did not sleep, we conducted a final experiment, where the four SLEEP instructions were replaced by the NOP-X instruction, which has no effect on the virtual CPU when executed, and has CPU and energy costs equal to the non-sleep instructions. The same experimental setup with a declining resource availability was used, the only difference being the replacement of the SLEEP instructions with NOP-X. Figure 3.4(b) compares the number of SLEEP and NOP-X instructions present and executed in the populations. In both cases the NOP-X instruction is significantly more plentiful than the SLEEP instructions. In fact the p-values for both are less than 0.0001 using the Wilcoxon rank-sum test. Selective pressures produced by this treatment favored doing nothing for 1 CPU cycle and paying a higher energy cost, over doing nothing for multiple CPU cycles and using 100 times less energy. Yet, even in the presence of this selective pressure, an adaptive resource-aware sleep/wake

behavior has evolved to a point where a majority of the organisms in a single population sleep at the same time.

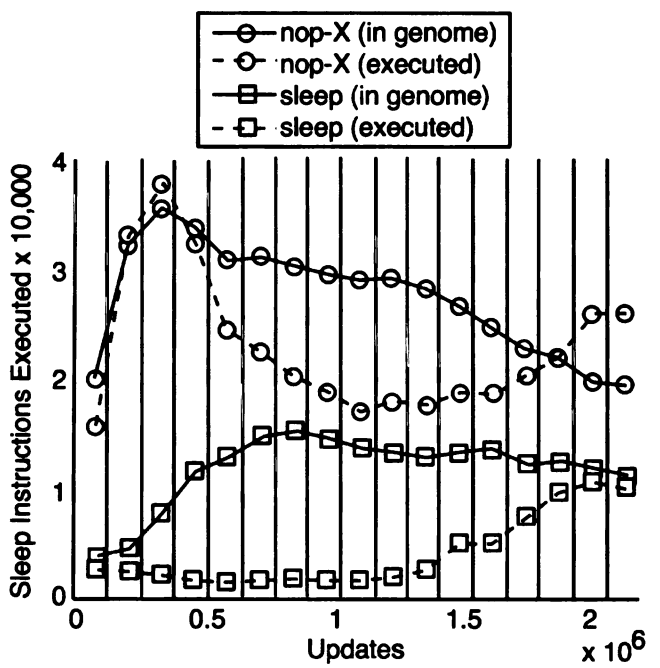
3.1.3 Conclusion

Revisiting the questions posed at the beginning of this section, we have shown that populations of digital organisms are capable of evolving resource-aware adaptive sleep/wake behavior in an environment where resource availability is periodic and declines over time. The organisms in these populations become highly active when the resource is available and sleep when it is not. This behavior evolves even though sleeping organisms are vulnerable to non-sleepers, and remained stable in a majority of the populations in our experiments. We also have seen evidence suggesting that the adage "early to bed, early to rise" describes an evolved behavior, as organisms maximize their probability of being rewarded for completing tasks. In addition, this behavior evolved even in the presence of a selective pressure not to sleep.

This work lays a foundation for several possible future studies. For example, it should be possible to evolve resource-aware behaviors using a pseudo-continous resource availability pattern rather than a binary resource. An example is a biologically inspired resource availability pattern that is relative to a modified trigonometric sine curve, simulating seasonal resource variations in addition to daily variations. To produce seasonally adaptive behavior the migration model, shown in Figure 3.5, could be used to produce a resource availability similar to that shown in Figure 3.6. In Figure 3.5, when an organism resides at a location near the celestial equator of a sphere, rotating around a single axis, the temporal resource variability it experiences is minimal. However, as an organism's descendants migrate north the temporal resource variability they experience gradually increases with higher latitude. This model of migration could be used to show that DE is capable of producing an adaptive behavior corresponding to the availability of a resource with seasonal variation. This model can also be extended by rotating the sphere on two axes instead of



(a)



(b)

Figure 3.4: (a) Attempted resource usage by organisms (resource activity) and resource availability vs. time for a typical 3-day interval. (b) A comparison of SLEEP instructions (squares) to inert NOP-X instructions (circles); solid lines indicate the frequency with which each instruction is found in the genome and dashed lines indicate the frequency at which they are executed.

just one.

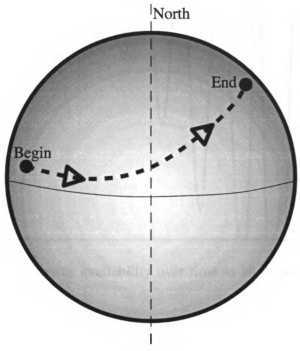


Figure 3.5: Model of organism migration starting at the equator where the resource is abundantly available and moving north to regions with less and less resource available for consumption.

In addition to the migration model, environments with added costs, instruction and environmental impairments, positive and negative reinforcement, and punishment could all be tested for effectiveness. In addition, environments encouraging predator/prey relationships could be examined for evidence of coexisting diurnal and nocturnal behaviors among organisms within the same population.

3.2 Cultivating Phototaxis

The goal of this set of experiments is to evolve a phototaxis behavior within Avida, then transition an evolved genome from Avida into a robot to observe the resulting behavior in hardware. To model the use of a battery we limit the total energy given to an organism

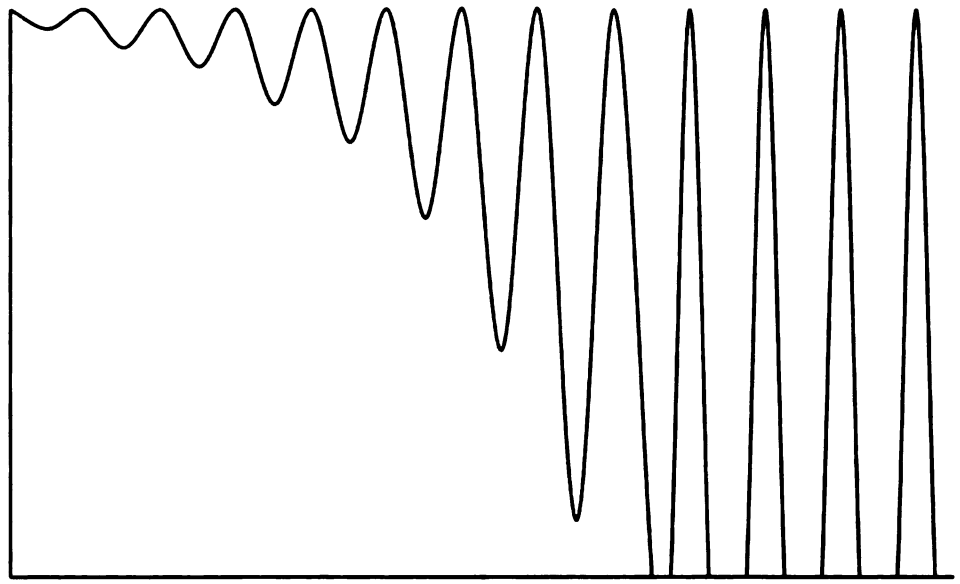


Figure 3.6: Mock daily resource availability over time as an organism's descendants migrate away from the equator.

to 10,000 energy units, and reward an offspring with additional energy proportional to its parent's remaining energy. Once the organism's energy is depleted, it can no longer execute instructions and is removed from the population. In addition, there is a 0.75% probability that the newly created genome is different from its parent. In our case study, we evolved mobility behaviors by placing a single organism into a 100×100 grid. This configuration provided the organisms with space in which to live and move without interference from other organisms, and eventually to evolve the desired behaviors.

3.2.1 Methods

Avida Extensions. In this study, an individual organism can interact with its environment in one of two ways: movement and sensing. An organism can move to a neighboring cell by executing the MOVE instruction. Besides movement, an organism can also interact with its environment through sensing. By executing a sense instruction an organism can gain information about the current environmental conditions. An example sense instruction is SENSE-CELL-DATA, shown in Figure 3.7. The SENSE-CELL-DATA instruction is similar to

a majority of Avida instructions in that its execution can be modified by a no-op instruction that immediately follows the SENSE-CELL-DATA instruction in a genome. By default, the SENSE-CELL-DATA instruction will place a sensed value into an organism's BX register. However, if a no-op instruction immediately follows the SENSE-CELL-DATA instruction then the sensed data will be placed in the register specified by the no-op.

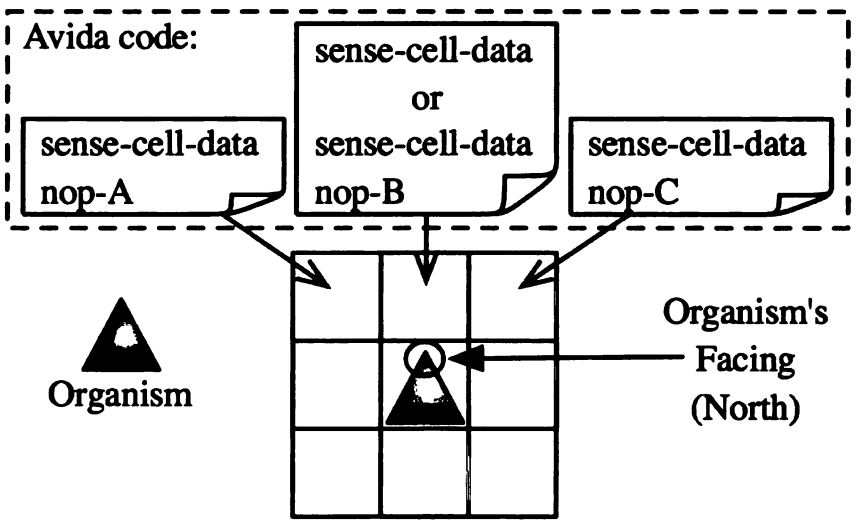


Figure 3.7: Depiction of SENSE-CELL-DATA

To increase similarity between the real-world and the Avida world, all of the movement and sensing instructions used in the following experiments have a higher energy cost than instructions that do not require external devices. This is done to discourage solutions from indiscriminately using these instructions and to loosely simulate the higher cost of using hardware that is external to the virtual CPU, such as sensors. A single base platform, an iRobot^{TM} Create, is modeled and used for comparison. However, separate treatments with various sensing capabilities are presented to demonstrate the flexibility of evolving behaviors for specific target robots.

Treatments. The goal of the case study is to evolve control software for robots, of varying capabilities, that allows them to search for and move to a light source. To do this, three different sensing capabilities are explored, illustrating the effectiveness of the methodology

in producing results for various embodied agents. The first of these sensing capabilities uses the COLLECT-CELL-DATA instruction. This instruction enables an organism to sense the intensity of a light at its current position. The second capability enables an organism to separately sense the intensity of a light source in three directions using the SENSE-CELL-DATA instruction described above. Specifically, the organism can sense the light intensity directly in front, in front and to the left, and in front and to the right of its current location. With the appropriate arithmetic instructions, these sensed measurements can be compared to determine in which direction the light is the most intense. The final sensing capability combines directional sensing with a rotate operation. Using the SENSE3-AND-ROTATE instruction, an organism can sense the light intensity within its environment and rotate to face the direction with the highest intensity. As will be shown, this level of functionality is highly suitable to the target task, and can be used as a building block for more complex evolved behaviors.

To encourage the evolution of object detection and location behaviors we apply selection pressures that reward organisms for efficiency in reaching the target, a simulated light bulb. Since instructions have different energy costs, variations in execution sequences can produce diverse energy consumption among organisms. Once an organism reaches a target, that organism's genome is replicated into another empty 100×100 grid. A bonus is given to the new organisms based on the amount of energy remaining in the organism. By applying selective pressures in this way, we reward individual organisms that can detect and move to the simulated light bulb in an efficient manner. In the following section we discuss the experimental results of these three treatments. Each treatment consists of 20 replicate runs, each containing 500 single-organism demes and lasting for 250,000 updates. Each individual is allowed to live for 1000 updates or until it is replaced. On average, an organism will execute a single instruction during an update.

3.2.2 Experimental Results

Initially we expected the evolved solutions that were exposed to the SENSE-CELL-DATA instruction to exhibit better performance than those exposed to the COLLECT-CELL-DATA instruction, due to the increased functionality of the former. However, this was not the case. Even though SENSE-CELL-DATA enables an organism to sense its environment in more detail, we never observed an evolved organism that used the full functionality of the instruction. We speculate that this effect arose due to the energy cost associated with executing this instruction and the simplicity of the environment. Specifically, the evolutionary process evolved organisms that could perform the target task with less than complete information about the environment. This result is encouraging to the extent that the evolutionary process did not need complete information of an organism's environment to complete the task. Actually, in both the COLLECT-CELL-DATA and SENSE-CELL-DATA treatments, evolved organisms exhibited an arc-like movement, where an organism would either move straight ahead if conditions were improving, or turn if not. In general, dominant genomes produced by these runs exhibited a loop, similar to the sample code shown in Table 3.4. The organism senses the environment and moves or rotates depending on whether or not the light intensity is increasing. Over the course of evolution, these loops were optimized to consume less and less energy, and we can see from Figure 3.8 that on average both treatments require a similar amount of time to detect and move to the simulated light source.

In addition to the total time required for an organism to move to the target location, we also tracked individual movments. Plotted in Figure 3.9 is the average number of organisms in the SENSE-CELL-DATA treatment populations that moved toward, away, and neutrally with respect to the location of the light source. A similar graph was also generated for the COLLECT-CELL-DATA treatment, but the results were not significantly different, and are not shown here. A notable aspect of this graph is that, in general, the same number of organisms are moving toward, away and neutrally with respect to the light source. This result is intuitive since often the evolved paths taken by an organism has an arc shape. This form of

Table 3.4: Sample portion of evolved genome using COLLECT-CELL-DATA

COLLECT-CELL-DATA NOP-C	sense intensity in cell store in CX
MOVE	move one cell
COLLECT-CELL-DATA	sense intensity in new cell store in BX
SWAP-STK	change stack; no effect
PUSH NOP-B	push BX on to stack no effect
IF-LESS	is BX <CX
ROTATE-RIGHT	then rotate right; else skip
ADD NOP-B	overwrite BX has no effect
MOV-HEAD	repeat

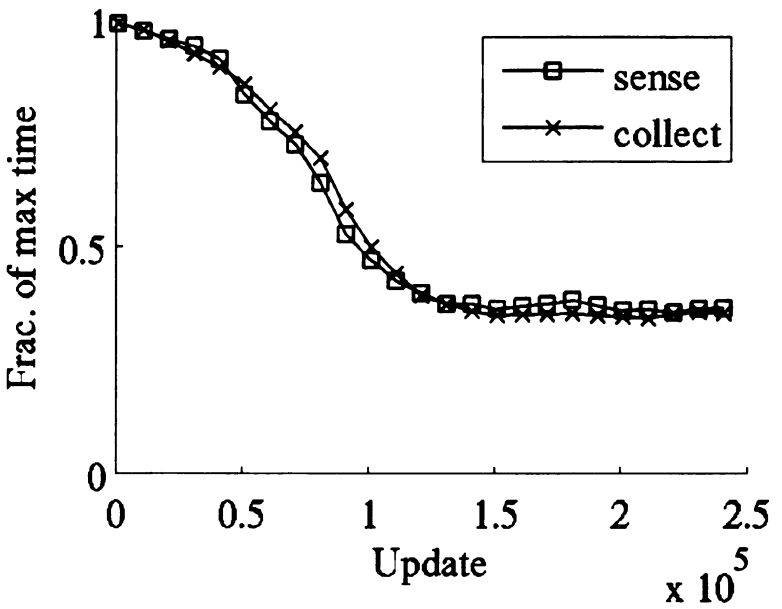


Figure 3.8: Average fraction of total allowed time required for a deme to be replicated in both the COLLECT-CELL-DATA and SENSE-CELL-DATA treatments.

mobility is dependent on an organism's memory of sensed values from multiple locations. In order for the organism to change its heading it must be subjected to a movement that either is neutral or away from the target. Thereby, the organism effectively depends on this type of "bad" movement to find the target, regardless of its adverse effect on energy consumption.

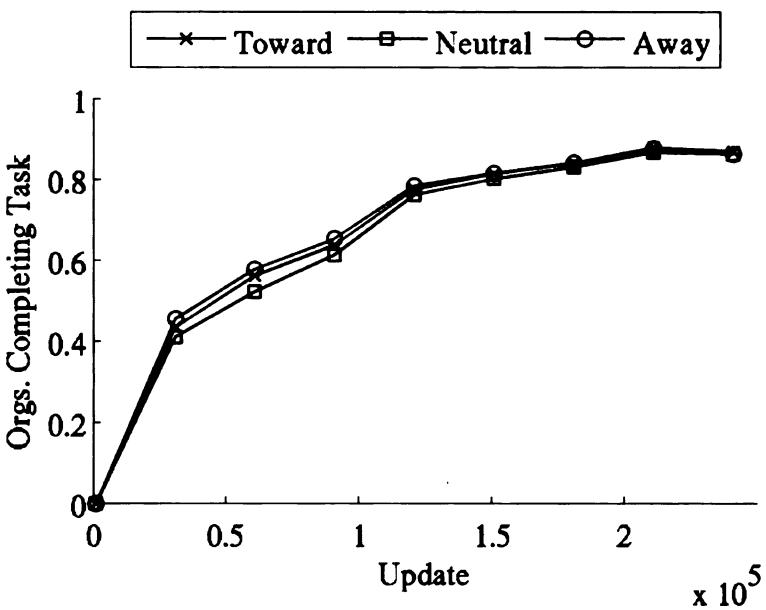


Figure 3.9: Average fraction of organisms in a population that have moved toward, away, and neutrally with respect to the light source in the SENSE-CELL-DATA treatment.

Building on the results seen in the two previous treatments and the prospect of evolving more complex behaviors in the future, we exposed populations of organisms to the SENSE3-AND-ROTATE instruction. As described above, the SENSE3-AND-ROTATE instruction automatically rotates an organism to face the cell with the highest measured light intensity. This is an example of a building block that a developer might consider adding to a system based on the problem domain. It also illustrates the flexibility of the methodology to handle various levels of target device capabilities. With the addition of the SENSE3-AND-ROTATE instruction the average fraction of the maximum time allowed to complete the task is drastically reduced (Wilcoxon rank sum test $\alpha = 0.01$ $p = 6.8 \times 10^{-8}$), as shown in Fig-

ure 3.10. Also, the average fraction of organisms in the population that move toward the light is significantly higher than those that move away (Wilcoxon rank sum test $\alpha = 0.01$ $p = 6.73 \times 10^{-8}$), as shown in Figure 3.11.

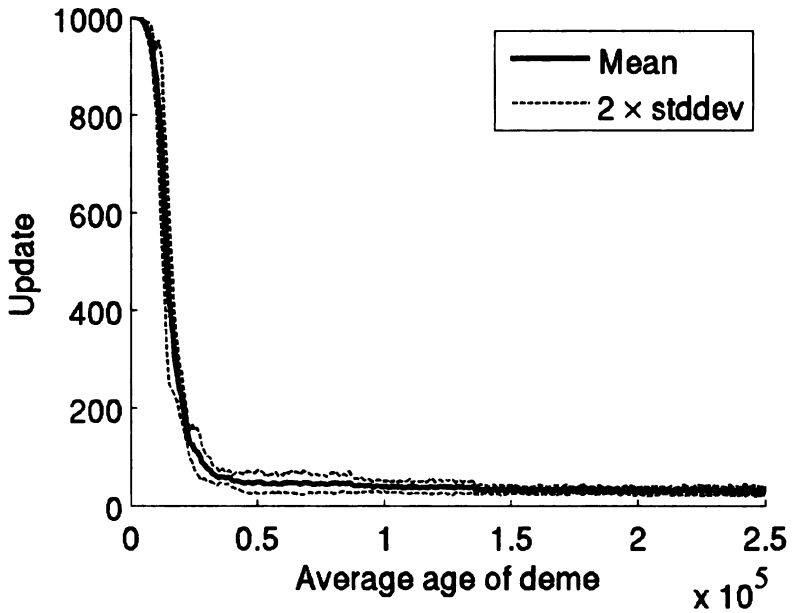


Figure 3.10: Average fraction of total allowed time required for a deme to be replicated in the SENSE3-AND-ROTATE treatment.

Table 3.5 shows a portion of an evolved genome, produced by the cultivation stage using the SENSE3-AND-ROTATE instruction. This partial genome is shorter and simpler than the partial genome shown in Table 3.4, but accomplishes the same task more efficiently. Even when energy costs for executing movement and sensing instructions are increased by an order of magnitude, this treatment still produces solutions capable of performing the task. However, the other two treatments are inhibited by such a substantial increase in energy cost, and are therefore more sensitive to individual instruction energy costs because of the increased number of instructions required to successfully complete the task.

Figure 3.12 shows a three-dimensional representation of the gradient used in the experiments. The black lines represent paths that organisms exposed to the SENSE3-AND-ROTATE instruction followed relative to the gradient. Depending on the organism's initial

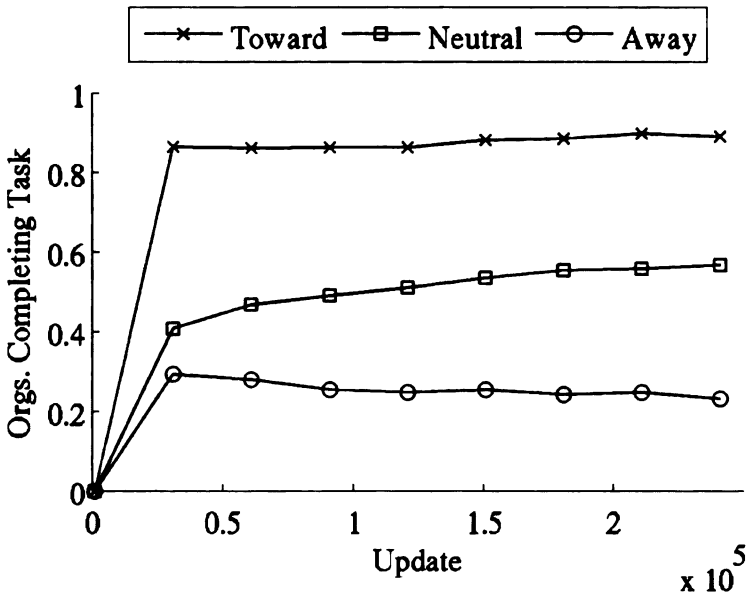


Figure 3.11: Average fraction of organisms in a population that moved toward, away, and neutrally with respect to the light source in the SENSE3-AND-ROTATE treatment.

Table 3.5: Portion of evolved genome exposed to the SENSE3-AND-ROTATE instruction

sense3-and-rotate	rotate in direction of highest intensity
sense3-and-rotate	
move	move straight
mov-head	repeat

facing the beginning of a path may display a few movements that allow the organism to orient itself to face the high point of the gradient, and then move in that direction. Since an organism is rewarded for energy conservation, the more direct a path, the better.

We translated several evolved genomes in to C code, compiled them, and loaded them onto an iRobot^{TM} Create robot. We then placed the robot in a room with a light source and filmed the resulting behavior. Four sample clips from one of these movies are shown in Figure 3.13. The images show a sequence of clips from the robot's initial position, orientation, its progress toward the light source, and then finally touching the light.

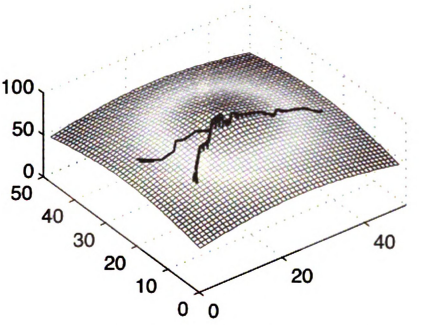
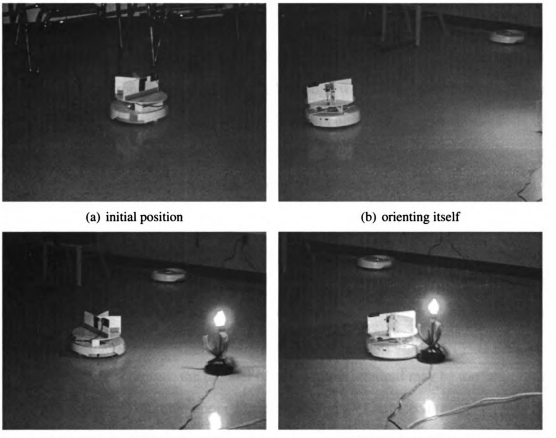


Figure 3.12: Example paths of dominant evolved genomes from the SENSE3-AND-ROTATE treatment, superimposed on the gradient used in the cultivation stage.

3.2.3 Related Work

The methodology described here is capable of producing autonomic behaviors in software systems, by using digital evolution to realize on a *developer's* high-level goals. Many traditional approaches to autonomic computing, where an *administrator's* goals guide the runtime behavior of the system, have been proposed, including those based on architectural models [173, 187], infrastructure for engineering emergent behavior [15], model-driven development [29], and design patterns [9]. In addition, the development of emergent systems [98] has been shown to produce robust, scalable, self-* systems [5, 96, 110]. Control over the population of agents [188], and methods to quantify their behaviors [194], show considerable promise. Our work complements these methods by using digital evolution to explore, as part of the software development process, possible solutions that realize autonomic behavior.

The case study presented in the paper is related to work in evolutionary robotics [144],



(a) initial position

(b) orienting itself

(c) moving toward higher intensity

(d) light source found

Figure 3.13: Clips from a movie that shows translated code produce by the SENSE3-AND-ROTATE treatment executing on an iRobot Create system.

which addresses the automatic generation of autonomic robots, and has been used to investigate questions in cognitive and neuroscience [81]. Work in this area has provided insight into the evolutionary conditions necessary for emergent communication [64], as well as communication protocols [123], multi-robot cooperation [55], and the concurrent design of a robot's morphology and its controller [118]. By combining autonomic computing with evolutionary computation, specifically evolutionary robotics, the proposed methodology provides a means to harness the power of evolution and apply it to the development of robust, scalable, self-* systems.

3.2.4 Conclusions and Future Directions

In this section, we have described a model that is capable of producing adaptive and autonomic behaviors. Through the use of this methodology we have demonstrated that various device capabilities can be successfully modeled and cultivated to produce solutions that, once translated, can execute directly on real-world hardware. We have also demonstrated how a developer can direct the process by providing high-level requirements for the evolved solution, and accommodate a change in the target platform's morphology by altering the capabilities modeled in the evolutionary process.

Employing evolution as the "designer" of software is not without its limitations. Well-crafted high-level goals and building blocks that allow evolution to produce desired results are needed. Also, while the virtual CPU architecture presented in this paper is capable of computing complex solutions, the number of instructions required may be too large for the evolutionary process to stumble upon. More capable virtual architectures are needed (and are under development). Despite these limitations, the proposed methodology has been shown to produce results acceptable for deployment of real-world devices.

This work could be extended into the production of evolved behaviors for swarm robots. However, the simulated environment used in this work does not map well to a physical environment when an organism is embodied. For example, the movement and sensing capabilities of an organism are discrete within Avida's cellular environment. As stated by Brooks in [25]:

> "These [cellular] representations are good for conducting computational experiments, and help uncover many fundamental issues. Unfortunately they do not shed light on all the problems which will be encountered when using physically embodied robots."

Extending Avida with a physics engine and evolving robots in the pseudo-continuous environment will lead to more realistic environmental simulation as well as produce solutions

that map well into the physical world. This extension can be used as a starting point for future researchers.

Chapter 4

Population Energy Management

In the previous chapter we focused on energy conservation within a single digital organism. In this chapter we extend this concept to an entire group of individuals, focusing on the collective ability of organisms to conserve energy while performing group-level tasks. Specifically, we investigate the evolution of a group's ability to conserve energy by managing its population and limiting the number of agents required to perform a given task.

Agent-based systems require population management to ensure the agents do not inhibit system functionality. For example, if the number of detectors in an artificial immune system is too small, then threats can go undetected, whereas if they are too numerous, the system can suffer from resource limitations. Dynamically adapting values associated with these management concerns (agent lifetime and number of agents) can directly affect a system's responsiveness, robustness, resiliency, and efficiency [23].

In this chapter we describe two experiments to promote the evolution of a self-organising, energy efficient group-level behavior. We begin with a discussion of related work (Section 4.1) and Avida mechanisms that facilitate the evolution of cooperative behavior (Section 4.2). Next, in Section 4.3, we describe experiments in the evolution of a self-regulating population. In these experiments we evolve a group of organisms to limit the number of individuals within a group while completing a task requiring multiple agents.

In Section 4.4 we present the results of experiments where we evolve a group to adaptively respond to a change in its environment, specifically, adapting the population size of a group to conserve energy depending on the current environmental conditions. Combined, these two experiments demonstrate that digital organisms can be evolved to both limit the number of individuals required to complete a task, as well as adapt to changing circumstances that require different population sizes.

4.1 Related Work

Population size regulation has been studied in the genetic algorithm literature [6, 62, 63]. Generally, however, the strategies used in these methods are static and may require global knowledge. For example, in [63], the lifetime of an individual has a fixed maximum, and global knowledge is used to keep the level of diversity within a population above a threshold. In our experiments an agent can alter its life span, and no global knowledge is used to promote diversity. While we do limit the maximum lifetime of an individual, this limit is extremely loose, greater than 50 times the maximum lifetime ever observed in our experiments.

Population management has also been addressed in systems-related fields [10, 33, 188]. Many approaches use *a priori* rules limiting population growth. For example, in [33] autonomous robots are evolved to capture a target while avoiding collisions. A predefined table specifies the replication rate, relative to the number of consecutively captured targets. In contrast, digital evolution involves no explicit control of the replication rate. Rather, natural selection drives a population to establish the number of individuals required to solve a problem. In addition, unlike the method described in [33], we do not allow information to be stored in the environment for use in stigmergic communication.

In addition, as described in Section 1.3, other studies of evolved and emergent behaviors have incorporated energy [65, 125] into their models. However, those works focus on

changes in a single individual when energy is considered. The work in the remainder of this chapter is concerned with group-level energy conservation and an evolved response to a changing environment.

4.2 Demes and Multilevel Selection

In some studies we treat all organisms as part of a single population [115, 116], in which case individual organisms compete against one another. However, in other cases, especially those involving the evolution of cooperative behavior, it is useful to subdivide the population and have groups of organisms compete. As shown at the bottom of Figure 4.1, a population of organisms can be subdivided into many sub-populations, called *demes*. In this work all demes have identical environments and initial configurations. However, an organism within a deme can interact only with other organisms in the same deme. Subdividing the population this way is akin to constructing "multicellular" organisms, enabling the selection of demes that perform group-level behaviors.

Multilevel selection [191] can be described as the application of natural selection at different granularities. Avida supports user defined multilevel selection, specifically, individual and deme-level selection. To enable deme-level selection, a deme is replicated when it satisfies a *deme-level predicate*, more generally thought of as a group-level behavior, such as flocking or consensus [104]. Upon deme replication, prior to creating an offspring deme, mutations are applied to the genome that was used to seed the parent deme. During this mutation process each instruction in the genome is subject to a 0.75% chance of being mutated to a random instruction. The newly created genome and its ancestral genomes make up the *germline* from which all seed organisms are produced. In contrast, other non-germline, or *somatic*, organisms play no role in deme replication, excluding predicate satisfaction. In addition to deme-level predicates, a deme's age is also used as a trigger for deme replication. This method allows for the bootstrapping of the evolutionary process by introducing

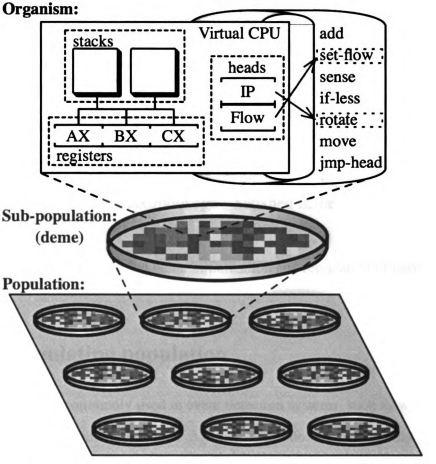


Figure 4.1: Population (bottom), sub-population (middle) and composition of a digital organism: genome (top right), virtual CPU (top left) with heads pointing to locations within the genome

mutations into a deme's germline.

Figure 4.2 depicts the initial injection of the ancestral organism into every deme, followed by both age- and predicate-based deme replication methods. While individual organisms within a deme are able to replicate, those replications do not involve mutations to the genome. Hence, all organisms within a deme are genetically identical. Floreano *et al.* have previously shown that this approach is effective in evolving cooperative behavior [64].

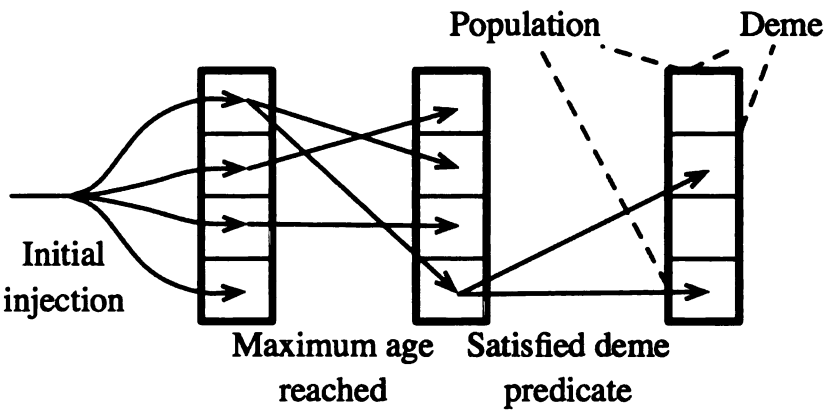


Figure 4.2: Example showing deme initialization and replication of germlines

4.3 Self-regulating population

Distributed agents are commonly used in event detection systems, such as wireless sensor networks and artificial immune systems. Agents can act both independently [67, 83] and cooperatively [168]. For example, in [67] agents independently detect the presence of a forest fire, collaborate to determine its perimeter, and notify local authorities. However, the QoS provided by this type of reconnaissance service, capable of surveying its environment and ascertaining strategic environmental features, is susceptible to agent under- and over-population. In general, the number of agents required for reconnaissance depends on the desired outcome. For example, if time is limited, more agents may be used to cover an area than when time is not an issue. However, if resource usage is also important, the number of agents may need to be restricted. Furthermore, some level of cooperation among agents

is required to effectively survey an environment and report events. In this section, we focus on the evolution of a cooperative deme-level reconnaissance task, specifically investigating the effects of a heritable energy trait on the evolution of this behavior in a multi-organism system.

4.3.1 Experimental Extensions and Setup

In addition to local computation and self-replication, a digital organism is also capable of inter-organism messaging, movement, and environmental sensing. Messaging functionality is provided by a BROADCAST instruction, which when executed collects the contents of two virtual CPU registers and transmits them in a single message to every organism within a user-defined radius. As an example, Figure 4.3 depicts three possible broadcast radii of an organism S placed in the center of a grid. If the organism's broadcast radius is set to 2, then every organism residing in a cell marked with a number less than or equal to 2 will receive a copy of a transmitted message. The results presented in Section 4.3.2 use a broadcast radius of 3, however, a broadcast radius of 1 was also tested and produced similar results. In addition to messaging, an organism can also move to a neighboring cell by executing the MOVE instruction. An organism will always move to the cell that it is facing. For example, if the organism S in Figure 4.3 is facing right and it executes a MOVE it will relocate to the cell marked with a "+1". An organism can change its facing by executing a ROTATE instruction. Upon birth, an organism initially faces its parent. Besides messaging and movement, an organism can also sense its local environment. The operation of the SENSE instruction will be discussed in Section 4.3.1.

The combination of local computation and environmental interaction effectively enables an organism to explore its environment and cooperate with others to perform a task. To encourage cooperative behavior an organism can be rewarded for completing an individual task that is a building block for a group-level behavior. For example, an organism could be rewarded for alerting its group of an important target when the group is surveying an

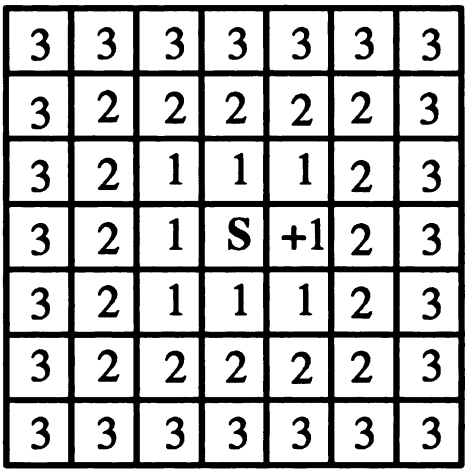


Figure 4.3: Example grid containing an organism S, and the cells reached by broadcasting with varying radii.

area. Once a rewarded task is completed, the organism receives an influx of energy and its metabolic rate is recalculated. By efficiently performing individual tasks an organism can increase its metabolic rate, giving it a competitive advantage. In addition, by decomposing a group-level behavior into individual building blocks, the Avida user can encourage the evolution of a complex cooperative behavior [18, 104].

In these experiments a population is divided into 100 independent demes, each consisting of 49 cells arranged in a 7×7 grid, as shown in Figure 4.4. Each cell within a deme is marked by an integer denoting the cell's contents: empty (-1), a "nest" (0), or a target (> 0). Each deme contains exactly one nest cell, located in its center, and one randomly located target cell; all other cells are empty. An organism can sense what type of cell it resides in by executing the COLLECT-CELL-DATA instruction, which reads the value stored in the cell into a register in the organism's virtual CPU. In the experiments described here a single deme-level predicate is used. To satisfy this predicate, a message containing the target cell's ID (a random positive integer stored in the target cell) must be received by an organism currently residing in the nest. Minimally, this predicate requires two organisms to cooperate; one to send the message and one to receive it. Upon satisfying of this predicate the deme is replicated, as shown in Figure 4.2.

-1	-1	-1	-1	-1	-1	-1
-1	>0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1

Figure 4.4: Deme setup with a nest (0), target (> 0), and empty (-1) cells.

To encourage the evolution of the desired behavior, two organism-level tasks are rewarded. The simplest task rewards an organism that enters the target cell, with an energy bonus equal to the baseline energy given to a seed organism (1000 energy units). Incorporating this task into the environment encourages organisms to forage for the target cell. However, this task does not require the organism to take any action or even have knowledge that it is in the target cell. To encourage active sensing and reporting of the target cell's ID, the second organism-level task rewards an organism for sending the target cell's ID in a message. However, before this task can be rewarded an organism must gain access to the target cell's ID either by finding the target cell (encouraged by the first task) and collecting its ID, or by receiving it in a message. After the organism has gained access to the target cell's ID, it must send the ID to an organism in order to receive a reward. Once this final step is completed the organism will receive a reward of 200 energy units. By performing these tasks an organism can increase its energy and gain a competitive advantage. However, it is conceivable that an organism could evolve to repeatedly complete either or both tasks. To discourage this type of hyperactivity, a limit is placed on the number of times an organism can receive a reward for each task. In addition, higher energy and virtual CPU cycle costs are assigned to all sensing, messaging, and movement instructions, mimicking the costs associated with performing these operations on physical hardware.

4.3.2 Experimental Results and Analysis

Evolved Foraging Behavior. The experiments described above produced demes capable of satisfying the deme-level predicate. Before evaluating the effects of various parameter settings on the evolutionary process, let us first describe a strategy that evolved frequently in our runs. We note that an organism cannot glean information about the location of the target cell from the environment unless it occupies that cell and hence, the only way an organism can find the target cell is to search for it by roaming about the environment. However, organisms evolved to take advantage of the constant location of the "nest" cell and the topology of the environment, as depicted in Figure 4.5. Specifically, through the use of the GET-CELL-XY instruction, which places the organism's current (x,y) coordinates in two of its registers, and the IF-EQU register comparison instruction, organisms repeatedly moved back and forth along the deme diagonal, where the x and y coordinates are equal. This oscillatory behavior enables an organism to move while remaining near and frequently entering the "nest" cell.

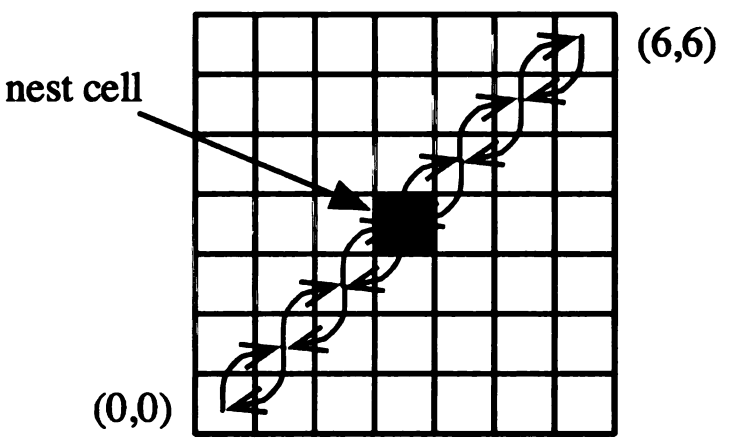


Figure 4.5: Example path resulting from organism moving back and forth on deme diagonal.

Varied Energy Transfer. To perform the following experiments we extended Avida to allow a percentage of a parent deme's energy to be passed to its offspring. The passing of energy allows it to be a heritable feature, thereby enabling selection based indirectly on

group-level energy efficiency. By varying the amount of energy passed to the offspring deme we are able to assess the effects of energy heritability on the evolution of a deme's ability to satisfy the deme-level predicate. We varied the amount of energy passed to the next generation in four different treatments: 0%, 1%, 5%, or 10%. Additional, higher levels of energy transfer were also tested, however, none was significantly different than the results observed in the 10% treatment. To measure the effect of energy transfer on the evolution of the development of a deme-level predicate, we compare each treatment based on the mean gestation time of a deme (time to complete deme-level task), and the mean number of organisms within a deme. We also use organism gestation time to evaluate the effects of energy transfer on the evolution of the deme-level task.

Figure 4.6 plots the effect of varying the percentage of energy transferred from the parent deme to the offspring on the mean gestation time of a deme. The plot shows a significant difference, at 50,000 updates and after, between the 0% treatment and all other treatments. For example, at 50,000 updates the Wilcoxon rank sum test for equal medians produces a p-value of 0.0025 for $\alpha = 0.001$. This plot suggests that as little as 1% energy transfer from parent to offspring can significantly increase a deme's ability to evolve a deme-level task, when compared to the 0% treatment. This result can be attributed to the fact that an organism injected into a deme in the 0% treatment is given the baseline amount of energy, which eliminates any energy advantage that could have been achieved by the parent deme, effectively slowing (but not stopping) the evolutionary process, as shown by the persistent downward slope in Figure 4.6. For example, if organisms in a deme increase their energy in the 0% treatment, then the deme will more likely be replicated. After replication, however, the energy level of the organisms in the offspring deme is reduced to the baseline, decreasing the deme's probability of replicating again. On the other hand, if energy is transferred to organisms in an offspring deme, the higher organism baseline energy level gives the deme a competitive advantage.

In addition to increasing the evolvablity of a system, a small transfer of energy can

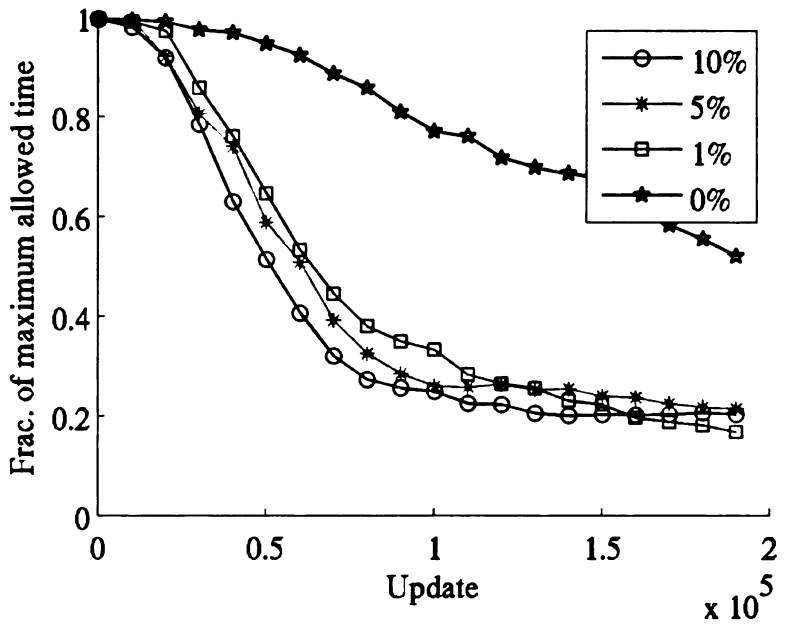


Figure 4.6: Average fraction of total possible time to complete a deme-level task using multiple energy transfer percentages. Results are mean of 30 runs.

also promote the evolution of a self-regulating population during the deme's development. Figure 4.7 plots the mean population size of demes in all four treatments. This plot reveals a mean increase in deme population size in the three non-zero treatments during the beginning of a run followed by a continual reduction after about the first quarter, eventually finishing below the 0% treatment. In contrast, the 0% treatment does not exhibit much variation in deme population size. However, the final resulting population size differences are not statistically significant.

Organisms in the 0% treatment do not perform individual tasks at the same level as organisms in the 1% treatment, as shown in Figure 4.8. However, the task completion statistics converge toward the end of both treatments, a behavior that is a byproduct of the deme replacement method and the decrease in deme gestation time. Specifically, the drop in task completion levels in the 1% treatment is caused by demes that are replaced before they perform a task. The lower levels of individual task completion in the 0% treatment are due to an absence of a selective pressure to complete these tasks and collect additional

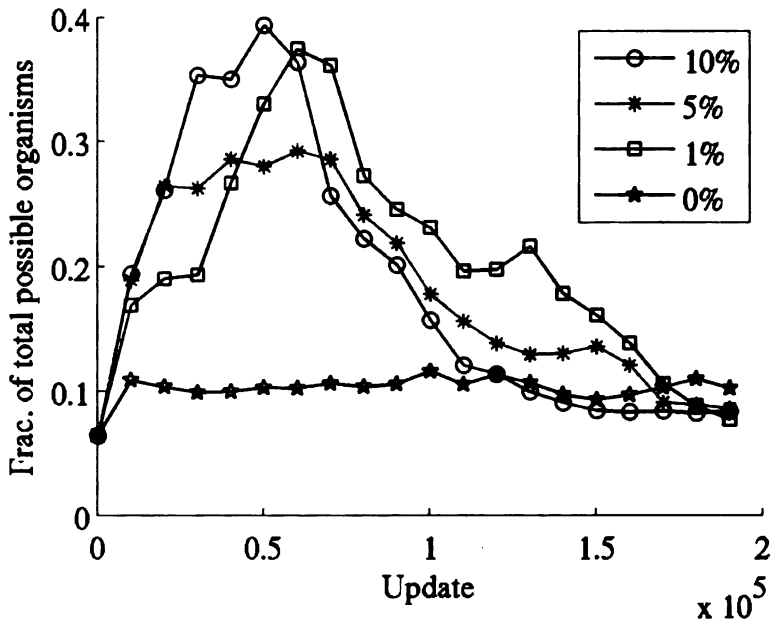


Figure 4.7: Average fraction of total possible organisms per deme using multiple energy transfer percentages. Results are average of 30 runs.

energy. In addition, since the organisms collect little additional energy, they are not able to increase the population in their deme above the level achievable with the baseline energy. However, even without a fluctuating population, the evolutionary process selects demes in the 0% treatment that satisfy the deme-level predicate, but this process requires more time than when energy is transfered, as seen in Figure 4.6.

The reduction in deme population size observed in the non-zero treatments in Figure 4.7 suggests that organisms have evolved in one of three ways. Either the organisms have (1) increased their level of cooperation, enabling them to satisfy the deme-level predicate more quickly, thereby reducing time for deme replication (supported by the decline in average deme gestation time shown in Figure 4.6), or (2) their replication rate has been slowed such that each organism reproduces less often, giving the group more time to satisfy the predicate before producing offspring, or (3) some combination of both. Figure 4.9 shows the mean gestation time of an organism for the 0% and 1% treatments. (The other non-zero treatments produced results similar to the 1% treatment and are omitted.) Error bars

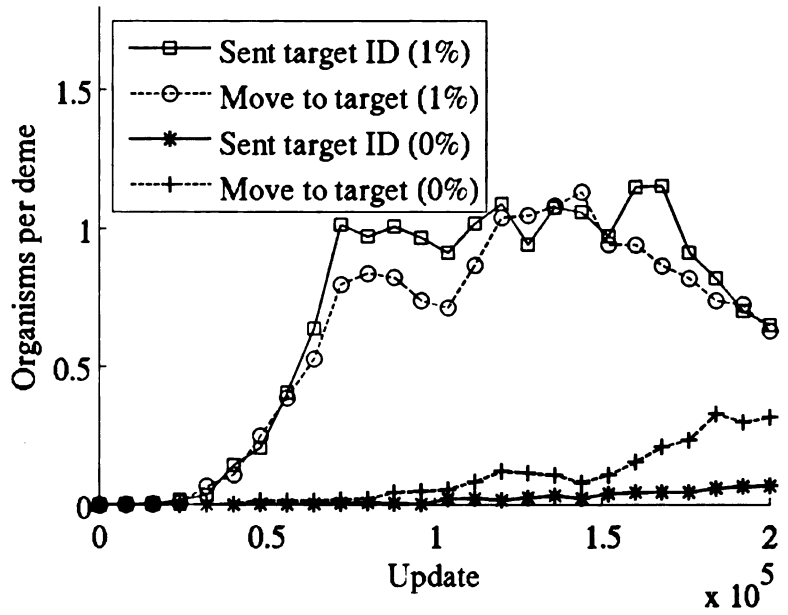


Figure 4.8: Average number of organisms in current demes who have performed either of the two individual tasks. Results are average of 30 runs.

are omitted from the figure because the two treatments are not statistically significantly different. We note that in both the 0% and 1% treatments, the mean organism gestation time increases over the duration of the run. This phenomenon occurred in all energy transfer levels tested. In contrast, the gestation time of Avida organisms typically decreases over time, as shown during the beginning of both treatments, because of selective pressures at the organism-level to become a more efficient self-replicator and produce more offspring. This result shows that this pressure can be overcome by performing selection at the deme-level.

Abundant Energy. In the previous treatments, the amount of energy an organism could gain during its lifetime was limited by a restriction on the number of times it could receive a reward for completing an individual task. To investigate the effects of abundant energy availability, we removed this limitation. Repeating the previous treatments with abundant energy we observed no significant differences in the results. Figure 4.10 displays the mean deme gestation time and total number of organisms per deme for the 0% and 1% treatments

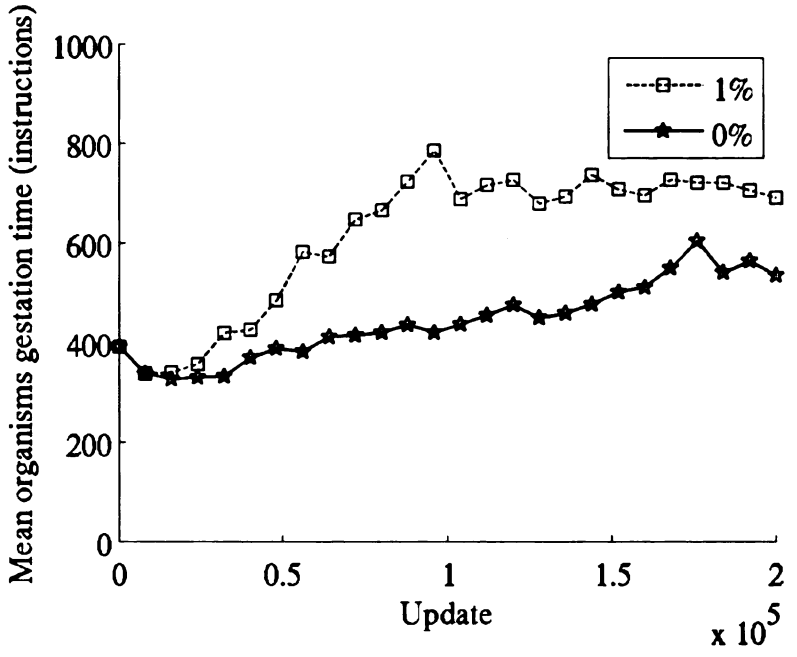


Figure 4.9: Mean of organism gestation times. Results are the average of 30 runs.

when energy accumulation is not limited. By inspecting Figure 4.10, we determine that the same pressures that caused the populations in the previous treatments to self-regulate are still present, even when energy is abundant. In addition, energy abundance does not significantly affect the gestation of individual organisms. These results suggest that energy abundance has little or no effect on the evolution of demes that satisfy the deme-level predicate. The minimal impact of energy abundance can be classified as a byproduct of diminishing returns: As an organism completes more tasks and accumulates additive energy rewards, it pays a higher energy cost per instruction because of its increased metabolic rate. Once the organism reaches the point where it costs more energy to perform a task than it receives in return, additional task completion begins to have a negative effect on the organism's metabolic rate. Therefore, the evolutionary process must balance diminishing returns with the selective pressure to accumulate additional energy by increasing an organism's gestation time.

The minimal effect of energy abundance on the evolution of a cooperative reconnaissance task suggests that deme-level selection is robust, at least in this case, to organism-

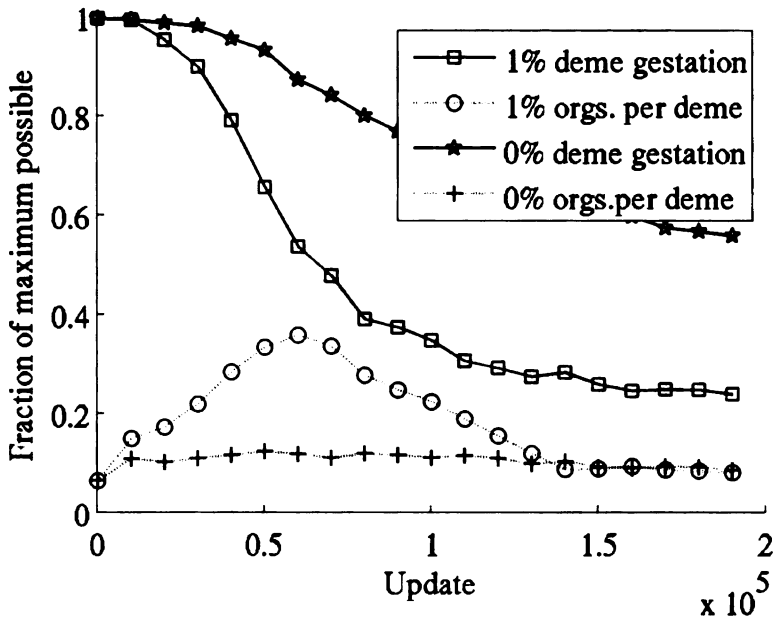


Figure 4.10: Fraction of total possible organisms per deme and fraction of maximum deme gestation time when energy is abundant and 0% or 1% of the parent deme's energy is transfered to the offspring. Results are representative of 30 runs.

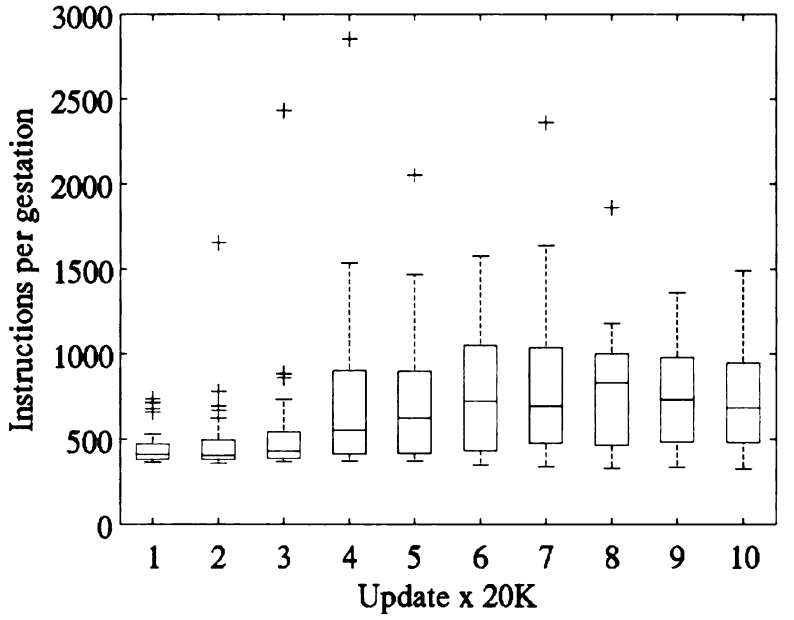


Figure 4.11: Average organism gestation time when energy is abundant and 1% of the parent deme's energy is transferred to the offspring. Results are representative of 30 runs.

level perturbation. In both the energy abundant and energy limited case, incorporating energy heritability into deme-level selection reduces the time required to evolve cooperative reconnaissance. In addition, the evolutionary process increases the quality of the solution by evolving a self-regulating population.

4.4 Population Adapting to Environmental Factors

Next we consider the evolution of an agent behavior to detect and mitigate "attacks" on cells in an energy efficient manner. We associated with each cell a certain amount of energy. When an organism enters a cell it acquires the energy within that cell and uses it to execute instructions. If an organism leaves the cell, its remaining energy is placed back into the cell and is unchanged until another organism enters the cell or the cell experiences an attack. An attack targets a single cell and acts as an energy sink. Attacks appear and are placed randomly in cells within a deme. When an attack is placed in a cell it draws down the energy within that cell, or an occupying organism, for the duration of the attack. (The duration and percentage of energy loss due to an attack are both fixed throughout a single run.) In the results discussed in Section 4.4.2, an attack's duration is set to 5 updates, and 1% of a cell's remaining energy is lost during every update during which the attack remains active. To conduct the study we added to Avida several new instructions related to communication, movement, environmental sensing, and changing their metabolism rate, as described below.

4.4.1 Experimental Extensions and Setup

Active Messaging. First, we defined instructions for *active messaging* [183], enabling organisms to alter the execution of other organisms by sending messages to them. The active messaging functionality is provided through two instructions, SEND-ALARM-LOCAL and SEND-ALARM-GLOBAL, which send an alarm signal to all organisms in neighboring

cells or all cells within a deme, respectively. As with any other instruction, these enter an organism's genome through random mutations during replication. An alarm message can be in one of two states, low or high, determined by the contents of the BX register in the sending organism: if the value stored in the register is even, then a low-state alarm message is sent, otherwise a high-state alarm is sent.

Both alarm sending instructions cause each non-sleeping, receiving organism to immediately move its instruction pointer from its current position to the location of an *alarm label* instruction, if present. An alarm label is a special purpose no-operation instruction used solely as a target for such an alarm-induced jump. The state of an alarm message is used to determine which alarm label is targeted for a jump. We defined two alarm label instructions, ALARM-LABEL-LOW and ALARM-LABEL-HIGH, which can be jumped to depending on the state of the received alarm. When an organism receives an alarm its genome is searched in the forward direction, from the instruction pointer's current position, until the correct alarm label is found or the search fails. If a corresponding alarm label is found, the receiving organism's instruction pointer is set to that location. If no corresponding alarm label is found, then no jump is performed.

Figure 4.12 depicts the execution of two neighboring organisms' genomes, initially organism *A* (top) is about to execute a SEND-ALARM-MSG-LOCAL instruction, which will send a high-state alarm to organism *B* (bottom). Once *A* executes the instruction, organism *B*'s execution jumps to the location of the ALARM-LABEL-HIGH instruction in its genome, skipping all instructions between the ROTATE-RIGHT and ALARM-LABEL-HIGH instructions. From this point on, both organisms execute their genomes sequentially until another alarm is received.

Movement and Rotation. As in experiments described earlier, an organism can move to a neighboring cell by executing a MOVE instruction with the new location is determined by its facing. An organism can change its facing by executing a rotate instruction. The

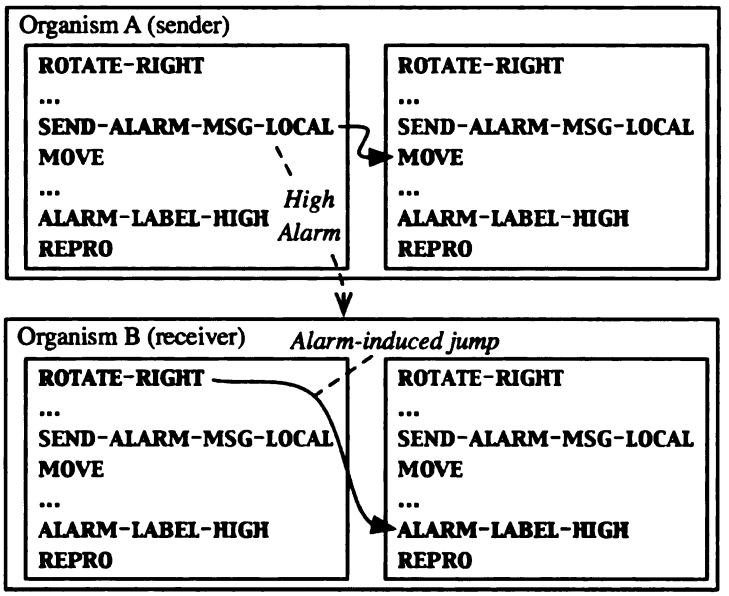


Figure 4.12: Example of the change in execution flow of organism *B* when it receives a high-state alarm from organism *A*.

ROTATE-RIGHT and ROTATE-LEFT instructions allow an organism to rotate one cell to the right or left, respectively. We expanded the instruction set to include three additional types of rotation. The ROTATE-UNOCCUPIED-CELL and ROTATE-OCCUPIED-CELL instructions, respectively, rotate an organism clockwise until a cell is found that is unoccupied or occupied; if no such cell is found the organism's facing remains unchanged. The third new rotate instruction, ROTATE-TO-ATTACK-CELL, rotates an organism clockwise until a cell containing an attack event is found; if no attack is found the organism's facing remains unchanged. For example, in Figure 4.13 organism *A* is initially facing to the left, but after executing a ROTATE-TO-ATTACK-CELL instruction it faces in the up direction.

Metabolic Rate. We also enabled an organism to have more direct control over its energy usage. In Chapter 3 we provided "sleep" instructions, and observed the evolution of a response to periodic resource availability. Here, we added two instructions, DOUBLE-ENERGY-USAGE and HALVE-ENERGY-USAGE, to enable an organism to change its execution priority by either increasing or decreasing its metabolic rate. When an organism

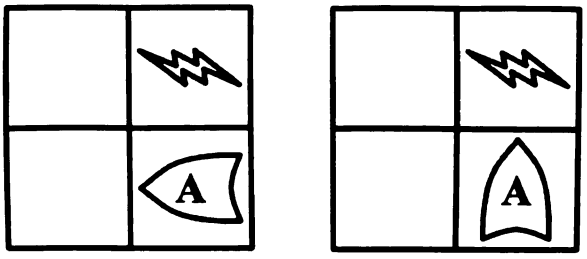


Figure 4.13: Example showing organism *A* (represented by a boolean OR gate) before (left) and after (right) executing the ROTATE-TO-ATTACK-CELL instruction. The attack is represented by the lighting bolt.

executes the DOUBLE-ENERGY-USAGE instructions, its metabolic rate is doubled, increasing the cycle speed of its virtual CPU. However, it will also pay double the energy cost per executed instruction. Moreover, if an organism increases its metabolic rate to a point where it can no longer pay an instruction's energy cost, then the organism dies. A third instruction, DEFAULT-ENERGY-USAGE, enables an organism to reset its energy usage to a default rate. In addition, upon replication both offspring organisms' energy usage return to the default level.

Dealing with attacks. The functionality to detect and quell an attack is available to an organism through special-purpose instructions. Specifically, we extended the Avida instruction set to include two conditional instructions that sense whether an organism either resides in or is facing a cell experiencing an attack. These two instructions, IF-CELL-UNDER-ATTACK and IF-FACED-CELL-UNDER-ATTACK, cause the next instruction in an organism's genome to be skipped if an attack is not present in the interrogated cell. In addition to these two conditional instructions, we also added two instructions that allow an organism to quell an attack. The KILL-ATTACK-IN-CELL and KILL-ATTACK-IN-FACED-CELL instructions, respectively, eliminate an attack in the organism's current cell, or the cell that it is facing. If an attack was present and mitigated, a 1 is written to the organism's BX register, otherwise a 0 is written. These four attack-specific instructions, along with all movement, active messaging, and sensing instructions, are assigned higher energy and

virtual CPU costs than other instructions. The increased costs facilitate the evolution of efficient and effective solutions for finding and quelling attacks.

In the experiments presented in the following sections, the rate of attacks during an attack period is randomly chosen to be either 0 or 5 attacks entering a deme per update. Every attack period lasts for 50 updates. Deme-level selection is used: a deme-level predicate is satisfied when a deme has successfully quelled 50% of all attacks that arrived during 5 consecutive attack periods. Once a deme has satisfied the deme-level predicate, it is replicated. At that point all of the deme's remaining energy is accumulated and divided equally among all cells in the offspring demes, minus a 5% deme replication decay. The transfer of remaining energy provides offspring of an energy efficient deme with a competitive advantage over less efficient demes. In addition, in the previous section we have shown that transferring energy in this manner can decrease the amount of time required to evolve cooperative behaviors [20].

4.4.2 Experimental Results and Analysis

We conducted a set of experiments to investigate whether digital evolution could solve the problem of quelling attacks while conserving energy, and if so, what behaviors it might produce. All the runs used Avida populations containing up to 4900 organisms, arranged in 100 demes, each consisting of a 7×7 torus of cells. When a deme is initialized, each cell is given $10,000$ energy units to be consumed by occupying organisms. A single seed organism is injected into the deme and begins to replicate, with mutations turned off (deme populations are homogeneous). If the population eventually satisfies the deme predicate, the deme will be replicated at the end of the current update. If a deme does not satisfy the predicate before it has experienced 500 updates, it will be replicated automatically because of its age. Upon deme replication, the germline may experience mutations. Specifically, there is a 0.75% chance that an instruction is mutated, and a 5% chance that an instruction is inserted and removed.

An Evolved Solution. Execution of the Avida runs described above produced populations that dynamically adjusted their size in order to quell attacks while conserving energy. Figure 4.14 demonstrates this behavior for a particular population. The figure plots the mean fraction of cells within a deme containing an organism during periods of attack and during periods of "calm" (no attacks). As shown, very early in the run the population evolves the ability to keep the population near its capacity when attacks are present, and it maintains this behavior during the entire run. After approximately 40,000 updates, the population has evolved the ability to adaptively reduce its population (to between 30% to 60% of its maximum size) during calm periods.

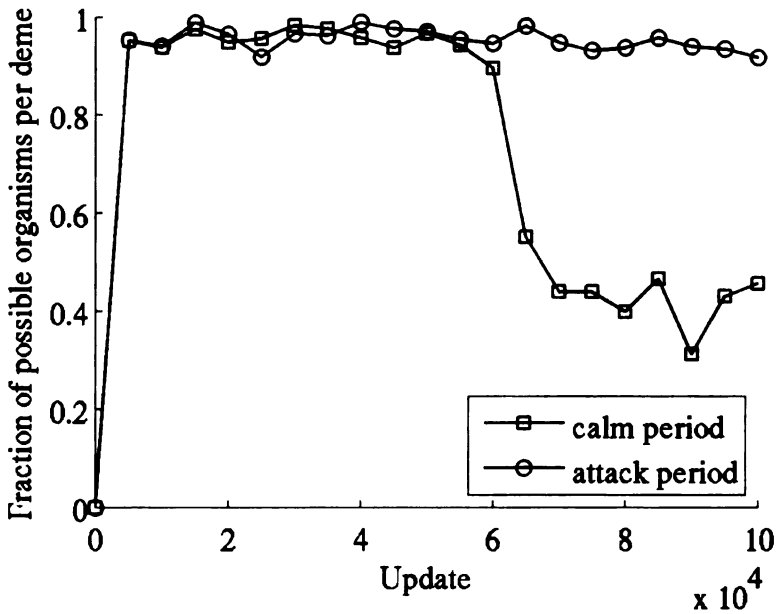


Figure 4.14: Mean fraction of total possible organisms within a deme when attacks are present and when they are not.

In addition to adaptively controlling the population, this particular population also achieves a mean attack mitigation rate of about 90%, as shown in Figure 4.15. Specifically, out of the 250 attacks occurring randomly during an attack period, about 225 are quelled. We emphasize that the only selective pressure is the deme-level predicate requiring a 50% mitigation rate. We do not provide any reward for low-level behaviors from

which the more complex cooperative behavior might emerge. Rather, the populations are entirely responsible for evolving their strategy.

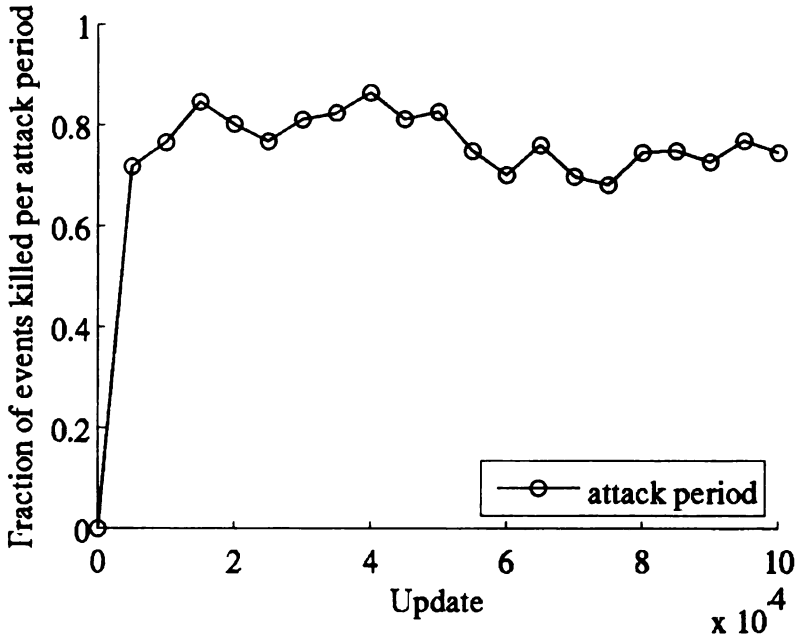


Figure 4.15: Average fraction of attacks quelled per attack period in a deme for a single run.

Genome Analysis. In order to understand how digital evolution went about solving this problem, we analyzed the dominant genome present at the end of the run, shown in Figure 4.16. This genome contains two separate pieces of code, one that executes when attacks are present and the other when they are not. The genome initially attempts to quell ("kill") an attack in the cell that it is facing. The remainder of the execution of this genome depends on the success or failure of this attempt.

If no attack was killed, then the BX register will contain 0 and a low-state alarm will be sent to neighboring nodes. However, this alarm will have no effect on neighboring organisms in the deme because the genome does not contain any ALARM-LABEL-LOW instructions. Therefore the remainder of the code in Box 1 of Figure 4.16 will be executed at both the sender and its neighbors. This code moves the organism and then executes a SHIFT-R instruction on the BX register. The right shift causes the organism's BX register to

either remain zero, or to become zero if an attack was previously killed. Then the organism enters into a low-energy (or "sleep") cycle, which costs the organism 1 energy unit and lasts for 30 virtual CPU cycles. Once the sleep cycle is over the organism replicates into the cell it is facing. Considering that demes are homogeneous, this method of replication allows the population of organisms to remain low. Specifically, because a parent organism never changes its current facing, the parent and offspring organism will always remain in the same row, column, or diagonal. Therefore, when either of them replicates, the population can grow only until that row, column or diagonal is full. After that point, all replications will replace existing organisms, causing the population size of the deme to remain constant.

On the other hand, if the attempt to kill an attack at the beginning of the genome was successful, then a high-state alarm is sent (since the value of BX is 1), and all organisms in the sender's neighborhood will jump to the top of Box 2 in Figure 4.16. If an organism is sleeping, it will remain sleeping, however its instruction pointer will be moved when it awakes in response to the most recently received alarm. The code within Box 2 can be divided into two parts: controlled spreading of alarm message, and racing to expand the population. The second and third lines in Box 2 control whether or not an organism that received an alarm message sends another alarm message. This piece of code will cause organisms that experience a high-state alarm message to propagate the message unless the organism's BX and CX registers contain the same value. As discussed before, the contents of an organism's BX register after it exits a sleep cycle will be zero. Therefore an organism that entered a sleep cycle will not send an alarm message when it wakes up, so out-of-date alarms will not be propagated through the network. The second portion of code in Box 2 doubles the execution rate of the organism and rotates it to face an attack if one is present in a neighboring cell. Finally, the organism replicates. After replication, the parent organism, which is facing the offspring, will kill any event in the offspring's cell, a rather distinct evolved parental behavior.

Figure 4.17 is a graphical depiction of the execution of two organisms during an at-

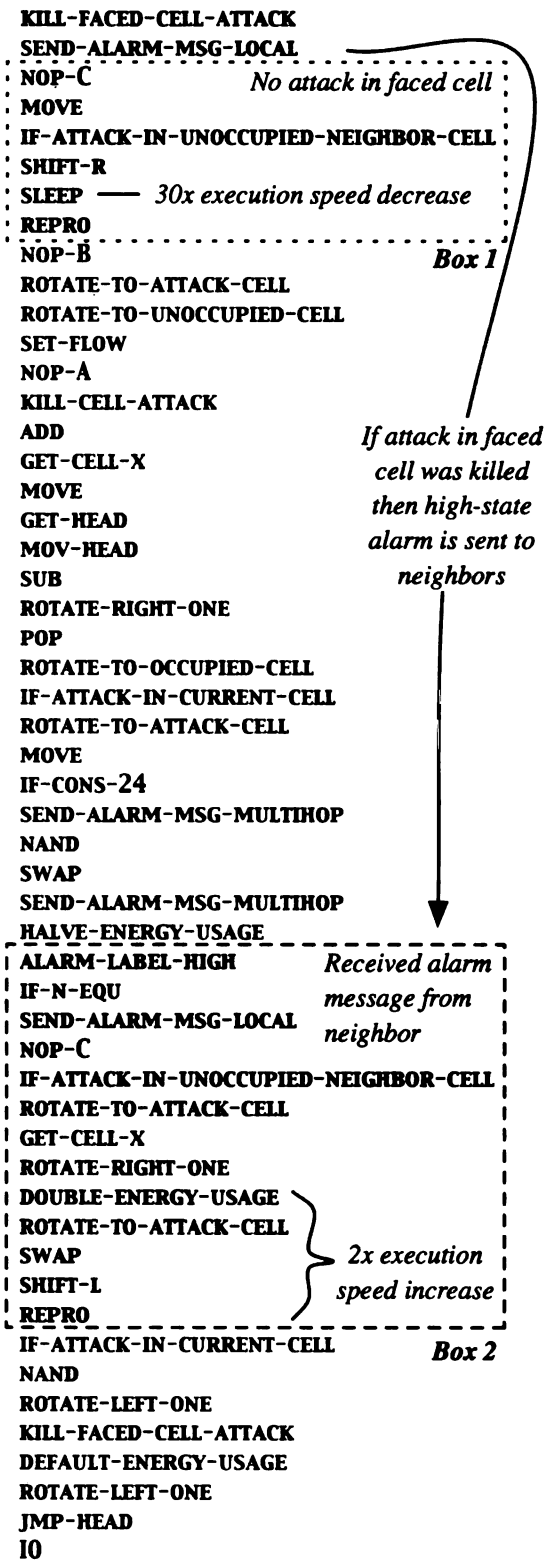


Figure 4.16: An evolved genome that executes differently depending on if a neighbor has sent a local high-state alarm message.

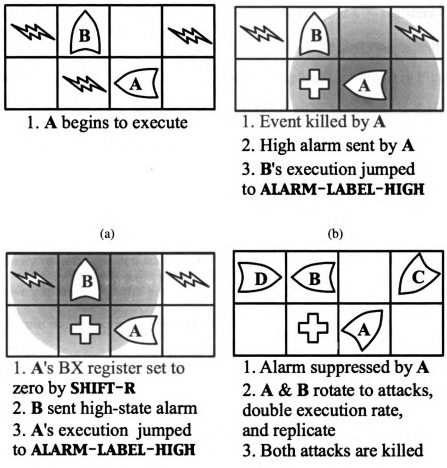


Figure 4.17: Sample population experiencing a period of attack. An organism is represented by a boolean OR gate where the point (or output) of the gate denotes the organism's facing. In addition, attacks, quelled attacks, and alarm messages are represented by lighting bolts, crosses, radially blended gray circle, respectively. Underlying each figure is a description of the individual executions represented.

tack period scenario. In Figure 4.17(a) organism *A* begins the execution of its genome, and organism *B* is assumed not to be sleeping. After organism *A* executes its first two instructions, the attack previously located in the bottom row has been quelled, *A* then sends a high-state alarm, and *B*'s instruction pointer has been jumped to the location of the ALARM-LABEL-HIGH instruction, as shown in Figure 4.17(b). After these instructions are executed, organism *A* shifts its BX register to the right, causing its contents to be changed from a 1 to a 0. At this point, organism *B* sends a high-state alarm message that jumps *A*'s instruction pointer to the location of the ALARM-LABEL-HIGH instruction. Now organism

A suppresses the sending of another alarm because its BX and CX registers contain the same value. Both organisms continue execution of instructions that cause them to rotate to face an attack, double their execution speed, and then replicate, as shown in Figure 4.17(d). Immediately after replicating, both organisms *A* and *B* quell the attack in their respective offspring's cell. By doubling its execution speed, an organism will pay a higher energy cost per instruction, however, it increases the probability of quelling an attack in its offspring's cell, since the time between rotating to face the attack and attempting to kill it is halved.

Energy Conservation. When accounting for each instruction's user-defined virtual CPU cycle cost, we calculate that the genome presented in Figure 4.16 can require as few as 9 cycles to execute during periods of attack and as many as 37 cycles to execute during calm periods. However, the lower bound on an organism's gestation time can be achieved only if a high-state alarm is received immediately after an organism replicates. This represents a 4-fold difference in individual organism gestation time depending on the current environment. Using these numbers to calculate the number of expected births during each period we predict a minimum of 1129 births during a calm period with 60% cell occupancy, and a maximum of 8167 births during an attack period. By examining Figure 4.18, we can see that the mean number of births during a calm period is slightly greater than expected. However, the mean number of births during attack periods is about half the expected total. This result suggests that organisms are replicating after executing about 18 instructions, which is 9 instructions more than required to execute the entire piece of code in the dashed box in Figure 4.16. We conclude that during attack periods, when high-state alarm messages are being sent, organisms are still conserving energy by sleeping when attacks are not present in their neighborhoods. On average, 1 in 5 organisms will sleep during a period of attack, conserving the energy in its cell.

Finally, we note that only 5 of the 20 runs achieved the level of success described above. Specifically, while most populations were similarly successful at quelling attacks,

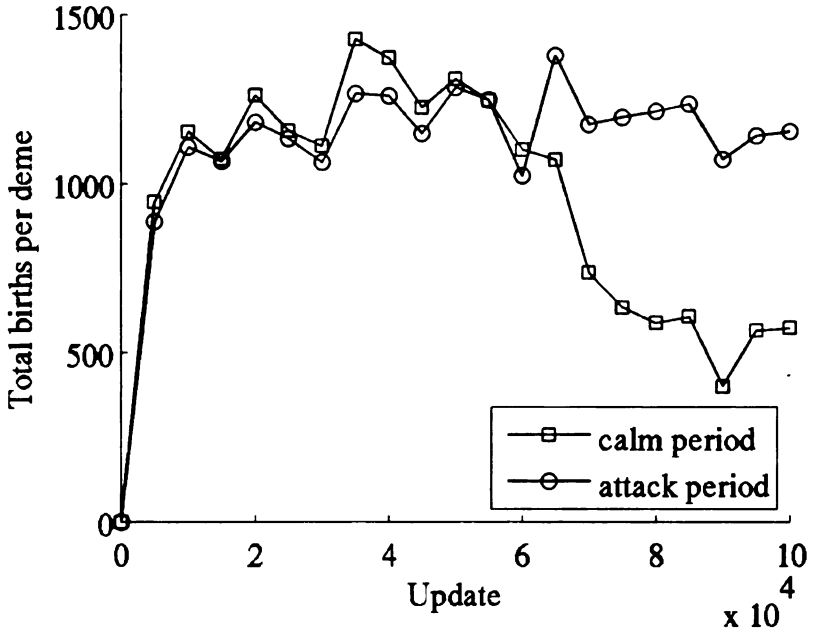


Figure 4.18: Mean number of births per deme in a single run.

only 5 were able to reduce the number of organisms significantly during calm periods. Of course, even a single successful population is sufficient for our purposes, as it can provide insight into the design of algorithms for agent-based distributed systems. However, from an evolutionary perspective this variation suggests that the selective pressures applied in this study might be improved, and in our ongoing studies we are exploring different pressures that may promote better energy efficiency and more effective self-regulation of population size.

4.5 Conclusion

In the first set of experiments described in this chapter, we have shown that a population can evolve to self-regulate its size when as little as 1% of the parent deme's total energy is transferred to the offspring demes. In addition, our experiments provide evidence that an increase in organism gestation time occurs when demes evolve to be more proficient at satisfying the deme-level predicate. In particular, an increase in the gestation time of

organisms allows a deme more time to satisfy the deme-level predicate with fewer total organisms, which translates into a more energy-efficient deme. Furthermore, we have shown that abundant resources have little effect on the evolution of this deme-level behavior.

In these experiments the evolutionary process is balancing opposing selective pressures: the pressure to decrease an organism's gestation time and the pressure to decrease a deme's gestation time. These two pressures are opposing because decreasing the gestation time of an organism will increase the number of births per deme, thereby increasing the amount of energy lost due to energy decay during replication. In contrast, a decrease in deme gestation time implies that fewer instructions are executed by its constituents, which translates into an energy savings. Since the deme-level predicate used in these experiments requires cooperation, evolution favors extending an organism's gestation time to allow more time to search for the target before replication occurs. These factors promote the natural selection of demes that satisfy the deme-level predicate while selecting against inefficient organisms, effectively encouraging deme-level efficiency.

In the second set of experiments, we have shown that it is possible to digitally evolve autonomous agents that response to attack fluctuations. Specifically, we described an evolved genome that self-regulates a population within a deme through the use of local active messaging. In addition, the genome adapts the execution speed of the host organism, depending on the current environmental circumstances. Finally, we showed that even during attack periods the evolved genome localizes the attack response, and conserves additional energy by putting approximately 1 in 5 organisms to sleep.

In an agent-based distributed system both individual lifetime and population size are important concerns for developers. Mismanagement of either of these two concerns can cause a disruption of a system's required QoS. Through the transfer of energy and deme-level selection, we have achieved a digital system that can effectively self-manage both of these concerns in addition to completing a desired task in an efficient manner.

Chapter 5

Quorum Sensing and Quenching

Building on the techniques employed to evolve adaptive individual and group behaviors in the previous chapters, we now focus on evolving density dependent, energy conserving behavior. This chapter describes experiments where digital organisms were evolved to perform a behavioral change once the population exceeded an evolved density threshold. We then attempt to disrupt the evolved behavior, and show that resistance to some of the disruptive methods is innate.

5.1 Background

An ability to self-organize is a key feature of an organism's development and behavior. For example, many biological organisms self-organize internally to form circulatory, nervous, respiratory, and metabolic systems. In addition, groups of organisms self-organize to find and build shelter, search for food, and coordinate attacks [30, 192]. Recently, analogs of these natural self-organizing behaviors have been studied for their application in computational systems [54]. For example, the evolved social behavior of ants [192] has led researchers to propose biomimemitic ant colony optimization [54], which has been used to design network routing protocols [9, 122]. However, due to interactions between multiple agents, many self-organizing systems are difficult to engineer and study. Furthermore,

adaptation in these systems adds to their complexity [85].

Quorum Sensing in Bacteria. Observations of behaviors in natural organisms provide useful insight into the complexity of self-organizing systems and the building blocks that underpin them. For example, although it was previously assumed that bacteria and other microorganisms rarely interact [155], in 1979 Nealson and Hastings found evidence that bacterial communities of *Vibrio fischeri* and *Vibrio harveyi* were able to perform a coordinated behavioral change, namely emitting light, when cell density rose above a threshold [142]. This type of density-based behavioral change is called *quorum sensing* (QS), and allows bacteria to coordinate gene expression under high cell density conditions [185].

Bacteria that participate in QS continuously release signaling molecules, called *autoinducers* (AIs) [143], which they can also detect with AI receptors. Under low-density conditions, AI molecules diffuse throughout the environment and go undetected by the bacteria. However, the level of AI increases with cell density and, if it exceeds a threshold, the detection mechanism in the bacteria causes the up regulation of the genes that produce AI molecules. This creates a positive feedback loop which greatly increases the level of AI in the environment. Once a receptor has been fully activated by a high concentration of AI, the activated receptor causes the up or down regulation of other genes in the bacteria. If the level of AI is relatively uniform throughout the environment, all of the bacteria that respond to the high level of AI will begin transcription of the same genes at approximately the same time, thereby changing the population's behavior once a quorum has been reached. The study of this evolved molecular communication system has enabled researchers to identify those genes whose transcription is density-dependent.

This discovery spawned a new branch of research to explore such interactions, determine whether they occur in other microorganisms [166], and assess consequences of these behaviors [47]. QS has since been observed in many species of bacteria, which use it for a variety of purposes, including secretion of digestive enzymes in the gastrointestinal

tract [26], bioluminescence and phototrophy in marine bacteria [21, 142], and polymer secretion to create or destroy a biofilm [140]. QS is also known to be closely related to more complex behaviors, such as aggregation into biofilms [79] and fruiting bodies [59]. For example, when confronted with starvation due to nutrient depletion, *Myxococcus xanthus* bacteria cooperate to form a stalk, enabling some cells to be carried as spores to new locations where conditions might be better.

In the case of pathogenic bacteria, such as *Salmonella* and *Staphylococcus*, QS has been linked to the coordinated release of toxins or other virulence factors [37, 46, 132], which attack a host's immune system [47]. For example, cholera-causing *Vibrio cholerae* adhere to an intestine wall of a host under low cell density. After a population increases and a quorum is reached, individual behavior changes, causing the bacteria to separate and spread, initiating an acute disease followed by a mass dispersal, enabling spread to other hosts [140]. The effectiveness of these attacks depends on large numbers of cooperating bacteria in order to overwhelm the host's immune response.

Improved understanding of QS has numerous scientific benefits [37]. Foremost, diseases caused by quorum-sensing bacteria might be treated with medications that inhibit this behavior (i.e., *quorum quenching*) [132], an approach that may have milder side effects than some antibiotics. For example, Davies et al. [42] showed that QS is essential to the development of biofilms in *Pseudomonas aeruginosa*, the primary pathogen observed in the lungs of people with cystic fibrosis. In addition, quorum quenching has been proposed as a possible treatment for methicillin-resistant *Staphylococcus aureus* [152], some strains of which are resistant to most traditional antibiotics [124]. Moreover, a deeper understanding of these interactions and their evolution may provide insight into the evolution of multicellularity itself.

In addition to numerous wet lab studies of QS and biofilm formation [26, 37, 42, 132, 142, 185], several researchers have constructed mathematical models that describe gene expression in QS bacteria [22, 162, 176]. These works differ from ours in that they use

P systems to model known gene expression mechanisms. In addition, Nadell et al. [141] recently simulated pairwise evolutionary competitions to investigate the production of extracellular polymeric substances (EPS) used in biofilm formation.

Artificial Quorum Sensing. Numerous computational systems also require a quorum to operate correctly. Most notably, algorithms for consensus, where members of a group agree on a particular course of action, represent a special-case of QS. Consensus algorithms support a variety of services in modern distributed computing systems, for example, distributed lock management [28] and ensuring the consistency of replicated components [14]. Distributed control systems also rely on consensus (and thus, quorum) in applications such as multi-vehicle control [161], where consensus-based algorithms are used to coordinate the movements of multiple autonomous vehicles. Additionally, sensor networks use quorum to perform clustering, where nodes are divided into groups in order to perform data aggregation [34, 193].

QS has also been proposed to coordinate behavior among multiple instances of computer worms [182]. QS can be used both to limit the scope of the worm and trigger its intended consequence. For example, Vogt *et al.* [182] discuss methods by which a worm could spread to an intended number of hosts and stop once a quorum is reached, ideally attracting less attention from system and network administrators because of the worm's lower profile. In addition, a QS worm could initiate a coordinated action when quorum has been reached, releasing what would amount to a "surprise attack" occurring throughout a network.

Knowledge of how relatively simple organisms cooperate to perform complex tasks may also be beneficial to the development of distributed computational systems that need to tolerate dynamic conditions and survive component failures as well as cyber-attacks. For example, collective behaviors among agents in an artificial immune system are essential to detecting and responding to potential threats, while nodes in sensor networks need

to implement complex distributed operations such as multicasting, gathering sensed data, and maintaining a network topology. Many traditional algorithms for solving these problems are brittle when deployed in dynamic environments, and several promising algorithms proposed recently are inspired by biology [9].

In this chapter we demonstrate the evolution of QS behavior in populations of self-replicating digital organisms. Specifically, we show that digital organisms are capable of evolving a strategy to collectively suppress self-replication when the population density reaches an evolved threshold. We describe the operation of an evolved genome exhibiting this behavior and analyze the collective behavior of a population that performs QS. We also show that the behavior scales to populations up to 400 times larger than those in which the behavior evolved. Additionally, we assess the ability of the evolutionary process to overcome communication impairments through an evolved resistance. We attempt to promote further resistance and demonstrate the effectiveness of these techniques in producing more robust organisms. This study (1) contributes to the understanding of QS and QQ, and (2) provides quantitative measurements of the effectiveness of impairments on an existing quorum. This study represents a first step in using artificial life, specifically digital organisms, to investigate the evolution, operation, and disruption of QS.

5.2 Quorum Sensing in Digital Organisms

5.2.1 Avida Extensions

Group Fitness. A deme's fitness is evaluated using Equation 5.1. After selection and prior to creating an offspring deme, mutations are applied to the genome that was used to seed the parent deme. During this mutation process each instruction in the genome is subject to a 0.75% chance of being mutated to a random instruction. In addition, there is a 5% change that a random instruction is inserted and deleted from a random location in the genome. The newly created genome seeds the offspring deme.

$$fitness_i = \begin{cases} 1 & \text{if } i \text{ is sterile,} \\ \frac{RemainingEnergy_i}{InitialEnergy_i} + 1 & \text{otherwise.} \end{cases} \quad (5.1)$$

In this work individual organisms within a deme are able to replicate, however those self-replications do not involve mutations to the genome. Hence, all organisms within a deme are genetically identical. Floreano et al. [64] have previously shown that this approach is effective in evolving cooperative behavior.

Avida Messaging and Interrupt Handling. Avida organisms can communicate by sending messages to one another. An Avida message consists of a single packet containing the values of two of the sending organism's registers. The send-msg instruction delivers a message to the organism residing in the currently *faced* cell. If the cell is unoccupied, then the message fails to be received. An organism can change its facing by executing one of several rotate instructions, discussed later.

In most prior studies using Avida messages, a receiving organism must explicitly *retrieve* the message from its input buffer in order to process it. However, we recently extended Avida with an interrupt model similar to the execution model of TinyOS [82], an operating system for sensor networks. In this model, depicted in Figure 5.1, an organism's main execution thread can be interrupted by a particular *event*, such as receiving a message. To enable context switching of this type, we introduced two instructions that denote the beginning (msg-handler) and end (end-handler) of an interrupt handler. We emphasize that these instructions have simply been added to the set of instructions available for mutation into an organism's genome. Whether they are used or not is solely a result of natural selection.

Figure 5.2 depicts the semantics of these instructions if they do enter the genome. When an organism receives a message, its genome is searched in the forward direction for the nearest instance of a msg-handler instruction. If none is found, the message is ignored.

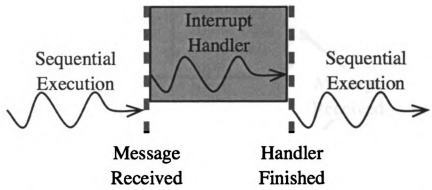


Figure 5.1: Context switch from sequential execution to an interrupt handler and back.

If a msg-handler instruction does exist, then the organism's context is saved, the contents of the message are placed in two of the organism's registers, and the instruction pointer is moved one instruction past the msg-handler instruction. The interrupt handler returns when an end-handler instruction is executed, which causes the interrupt context to be flushed from the organism's virtual CPU and the original context to be restored. If no end-handler instruction exists, then execution continues sequentially through the genome. Lastly, if a msg-handler instruction is encountered during normal sequential execution, the handler code is skipped and execution jumps past the next end-handler instruction. If no end-handler instruction exists then the jump is not taken.

In this work an interrupt handler cannot be preempted; therefore, all interrupts are handled atomically. Specifically, if a message is received while an organism is interrupted, the message is queued until the handler has finished, at which time the handler is re-entered and the next message in the queue is processed. The original context is restored only when all received messages have been processed. Furthermore, the incoming message buffer is limited to 20 messages. Messages received when the buffer is full are dropped. As we shall see in Section 5.2.2, the evolutionary process exploited this property in order to produce quorum sensing behavior.

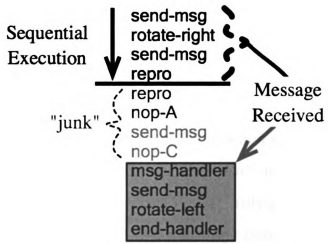


Figure 5.2: Sample genome containing a single interrupt handler. During sequential execution the first four instruction of this genome are executed. If a message is received the organism's IP is jumped into the interrupt handler. This example also contains "junk" code that will not be executed unless the genome is mutated.

5.2.2 Experimental Setup and Results

In this section we demonstrate that the digital evolution system as described above is capable of producing deme-level populations that exhibit QS. Specifically, we observed Avida populations that evolved a group communication behavior to inhibit self-replication once a deme has reached a density threshold. Moreover, the threshold itself was not specified *a priori*, but rather was an evolved characteristic of the population. In addition, we will show that this behavior arises under different initial conditions and is scalable to demes up to 400 times larger than the demes in which the behavior evolved.

Experimental Setup. In the experiments described below, multiple populations are divided into 400 demes. Each deme's topology is a 5×5 torus that is seeded with a single organism at the beginning of each competition period. A competition period lasts for 20 updates, where the average organism in the entire population will probabilistically execute 50 instructions per update. After each competition period, each deme's fitness is evaluated using Equation 5.1, and individual deme germlines are selected, mutated, and used to seed demes in the next competition period. Each seed organism is placed and rotated randomly,

so that its initial position and facing cannot be "learned" through the evolutionary process. In addition, deme-level populations are *well-mixed*, meaning that during replication, the offspring is placed in a random cell within a deme, with a preference for empty cells. Therefore, a population must be full for an organism to be overwritten.

This study focuses on the evolution of a digital organism's "gene" (instruction) regulation mechanism, specifically the change in an organism's instruction expression under low and high density conditions. To facilitate genome analysis, given in Section 3.4, we use an instruction set that is includes the send-msg, msg-handler, end-handler, and various rotate instructions described below. Additionally, four different no-operation instructions and the repro instruction are provided. This instruction set contains fewer instructions than previous Avida studies [116]. Other instructions could be included in the set of instructions available for mutation. However, since we are not applying any additional selective pressures, organism behaviors will evolve in accordance to the deme-level fitness function.

Treatments. Our initial experiments are divided into three treatments based on available rotation methods. There are three types: Single step, Labeled, and Neighbor-based. Single step rotations enable an organism to rotate one cell to its left or right by executing the rotate-right or rotate-left instruction, respectively. The rotate-label instruction enables an organism to rotate to a direction specified by a sequence of subsequent nop instructions, or *label*. (An explanation of labels and nop-modifiable instructions can be found in [149].) Lastly, the instructions rotate-occupied-neighbor and rotate-unoccupied-neighbor perform neighbor-based rotations. These instructions will rotate an organism to a neighbor cell that is occupied or unoccupied, respectively. Each treatment consists of 20 runs with each run lasting for 2500 deme competition periods (generations). Within a given run, the evolutionary process has access to only one of the three rotations methods, simplifying intra-treatment comparisons. In addition, we compare results from all treatment, however, all comparisons are limited to those runs that exhibit the desired overall behavior, namely

population control. Out of the 20 runs in each treatment, we observed 15 runs in the Single step treatment and 14 runs in both the Label and Neighbor treatments that exhibited this behavior, only those runs are used in the following discussion.

Energy Conservation. We begin by considering the amount of energy conserved per deme for the three different treatments. This value alone determines the fitness of the deme. We note that there are only two ways for a deme to lose energy: the deme's constituents use energy when executing instructions, and the energy remaining in an organism is purged when the organism is replaced. Since every instruction has the same energy cost, different instruction execution sequences of the same length all have the same explicit energy cost. Therefore, the only way organisms in a deme can reduce their total energy usage is to limit self-replication.

Figure 5.3 displays the average fraction of energy remaining within a deme at the end of a competition period. Since Figure 5.3 shows an increase in energy conserved per deme in all treatments, it can be assumed that the number of births per deme is declining over evolutionary time. This assumption is confirmed by Figure 5.4, which shows the average number of births per deme. For all three treatments, the number declines to approximately 30-35 births per deme.

Group Behavior. QS behaviors found in natural organisms exhibit two key features. First, a density based change in behavior can be observed, and secondly, after a threshold has been reached a positive feedback loop is created that causes more AI to be released. In this paper messages are analogous to AIs, and alternate gene (instruction) activation can be realized through the evolution of an interrupt handler. We define organism density as the number of organisms per cell; therefore, a density of 1.0 can be achieved only if every cell in the environment contains an organism. Figure 5.5 plots the average organism density per deme over evolutionary time. Both the Label and Neighbor rotation treatments exhibit a slight, yet steady decline in organism density. However, the Single rotation treatment

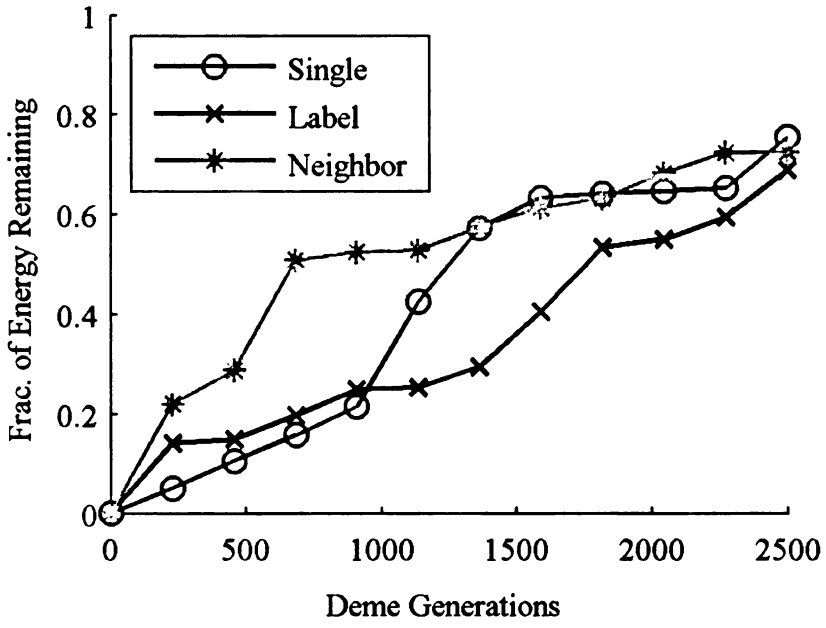


Figure 5.3: Mean fraction of energy remaining per deme over deme generations for each treatment.

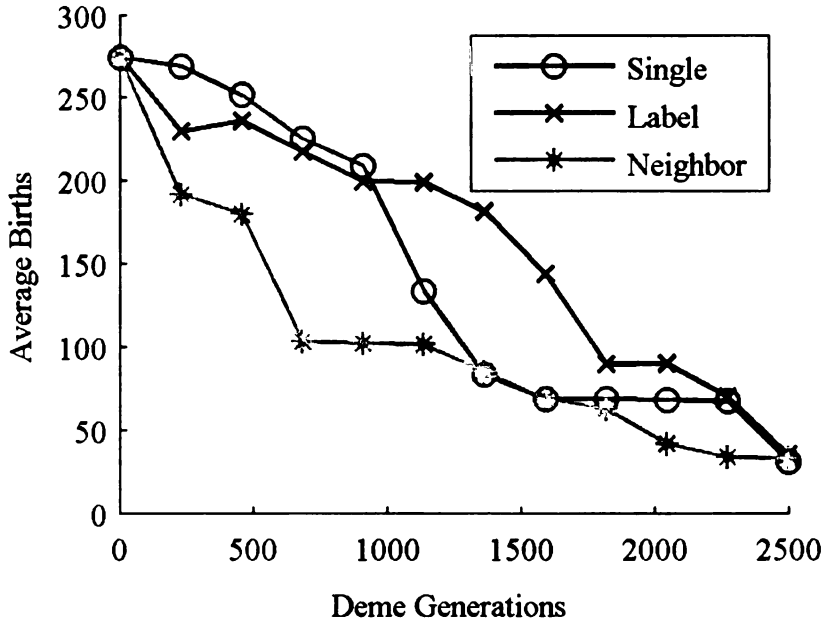


Figure 5.4: Average number of births per deme for all three treatments.

displays a larger decline in organism density, reaching a value of approximately 0.9. The Single rotation instructions provided organisms with the ability to easily send a message and then rotate one cell to the left or right. Performing this basic behavior in the Label treatment requires a much longer sequence of instructions and is therefore less likely to evolve. In addition, this basic send and rotate strategy is not possible in the Neighbor treatment without additional information about the organism's current neighborhood. In the remainder of this paper we focus only on results produced in the Single rotation treatment.

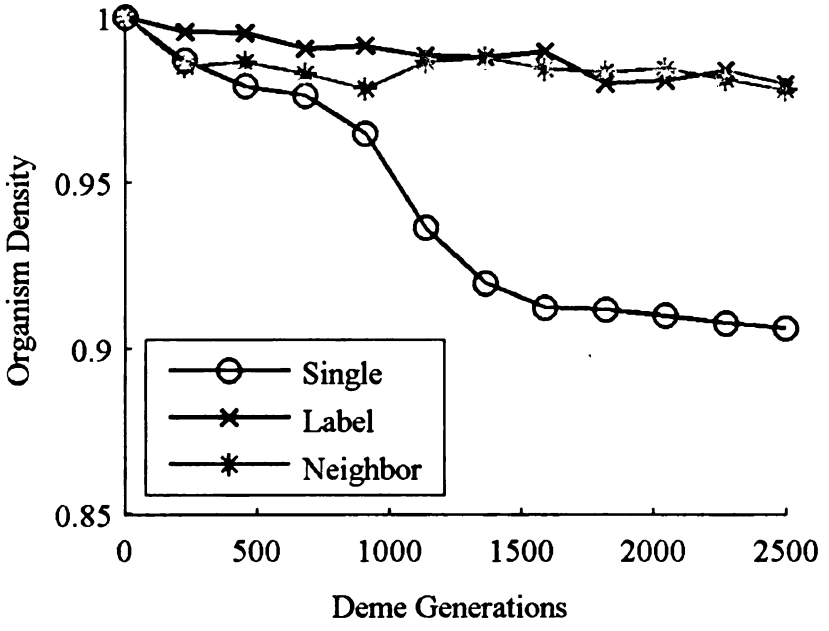


Figure 5.5: Average organism density per deme.

We focus on the most abundant, or dominant, genomes produced by runs. These dominant genomes are extracted from each run at its conclusion, and a replicate of each is used to seed 400 demes which are then run for one competition period. While these demes are executing, we record for each deme the organism density, total births, and number of organisms running in an interrupted state. Figure 5.6 shows graphically the correlation between organism density and total births per deme averaged over all dominant genomes. As seen in Figure 5.6, after update 4, both the organism density and total births per deme plateau. Therefore, deme-wide self-replication behavior, which is present before update 4, is sup-

pressed when the organism density reaches approximately 0.8, demonstrating a quorum has been reached.

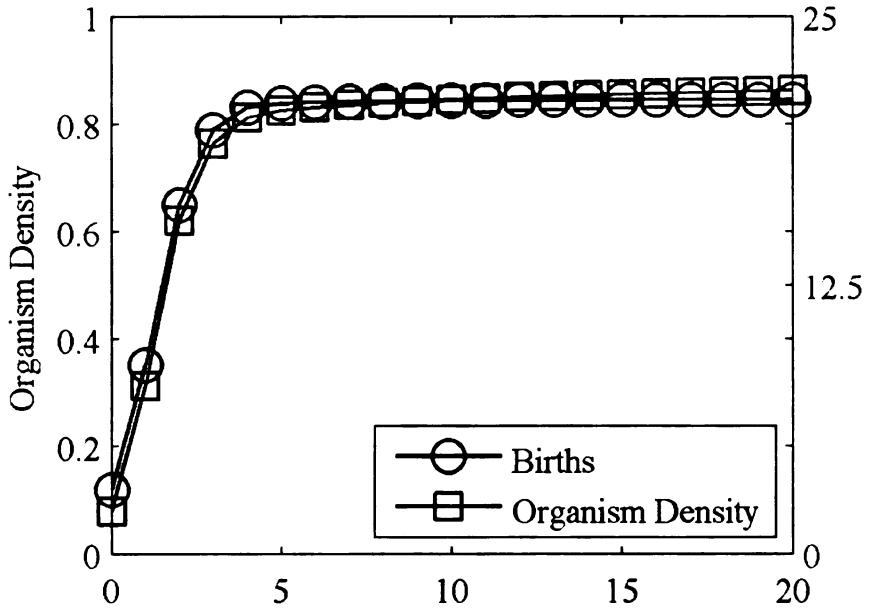


Figure 5.6: Organism density and total births per deme for dominate organisms.

To this point we have not discussed methods by which the organisms implement quorum sensing. Figure 5.7 plots the average number of organisms per deme that are executing in an interrupt handler. This curve closely mirrors the curves in Figure 5.6, providing an indication that interrupts are being used to suppress organism self-replication. In addition, if we disable messaging, therefore organisms cannot become interrupted, the average organism density within a deme quickly increases to 1.0 and all demes die out due to energy depletion. This provides further evidence that interrupt-causing messages are an important feature of the evolved genomes. Now let us focus on the genome of an organism that realizes this behavior.

Genome Analysis. Here, we focus on the dominant genome that produced the lowest average organism density of 0.67. An organism containing this dominant genome, shown in Figure 5.8, will execute the first 16 instructions in the genome before it replicates, as

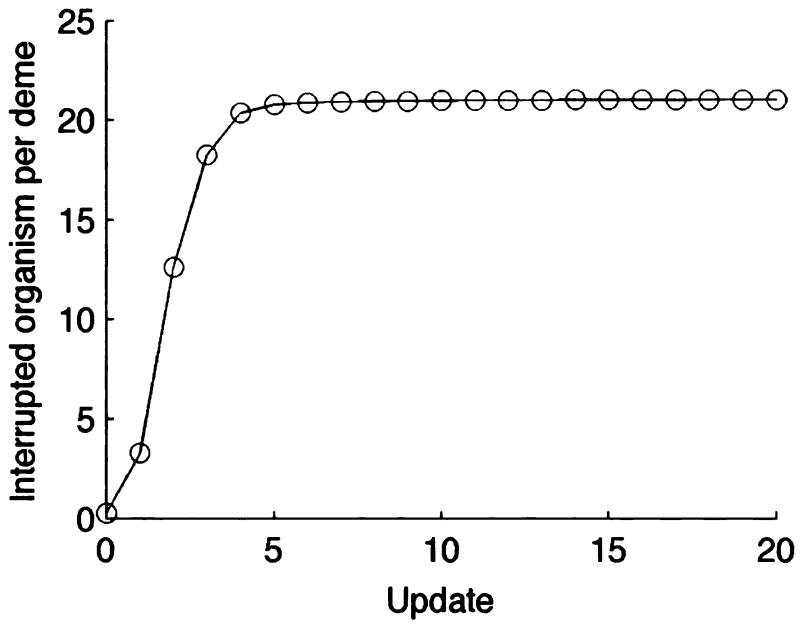


Figure 5.7: Average number of organisms interrupted per deme.

denoted by the black line to the left of the genome. During this sequence the organism sends a single message to its initially faced cell and the two neighboring cells to its left, finally replicating while facing the cell one rotation to the left of its initial facing. In short, the organism will send 3 messages for every 16 executed instructions during normal execution and then replicate. Upon replication the organism's state is reset, causing the genome to be processed from the beginning.

Since genomes are executed in a cyclic manner, this sequence will be repeated until the organism either runs out of energy, is replaced by an offspring of another organism, or is interrupted. If interrupted, the current context of this organism is saved and the interrupt-causing message is processed in the interrupt handler, denoted by the red boxes at the bottom (beginning of handler) and top (end of handler) of the genome in Figure 5.8. While interrupted, the organism will send one message to the cell it currently faces and two messages to the cell left of its initial facing. Moreover, the organism will remain in the interrupt handler, and hence will not replicate, until all received messages have been processed. Furthermore, for every entrance into the interrupt handler caused by receiving a message, the

organism will produce three messages. Hence execution of the interrupt handler produces a positive feedback loop where the level of messaging (AI) in the system is increased, thus *tripling* the chances than an organism will become interrupted. Therefore, the expression of this individual behavior in a dense group of digital organisms will cause an organism to remain in a state of perpetual interruption and never self-replicate.

Experiments with Larger Demes. To determine whether this QS behavior is truly density based we seeded demes several times larger than the original 5×5 demes in which the behavior evolved. Table 5.1 provides a list of deme sizes tested, their scale relative to a 5×5 deme, total number of demes per run, and total number of demes used to generate averages plotted in Figure 5.9. Note that only 15 of the 20 runs evolved population control behavior, so only those dominants were used, producing the values in column 4 of Table 5.1. Each deme is again allowed to execute for a single competition period and the average organism density for all three scalings over time is shown in Figure 5.9. The final average densities produced for all three scalings are insignificantly different. Therefore, the evolved dominant genomes exhibit a quorum sensing behavior that suppresses organism self-replication under multiple scalings. In addition, a quorum is reached at similar organism densities.

Table 5.1: Deme size comparision

deme size	$N\times$ larger	# demes	total demes
5×5	1	400	6000
25×25	25	20	300
50×50	100	10	150

Finally, we seeded a single 100×100 deme with an organism containing the genome in Figure 5.8. We then allowed the deme to execute for one competition period, and we tracked the constituent organisms' execution behavior. Figure 5.10 shows snapshots of the resulting behavior, where each (x, y) coordinate corresponds to a single cell in the 100×100 deme. A cell is colored according to the current activity within that cell. If a cell does not

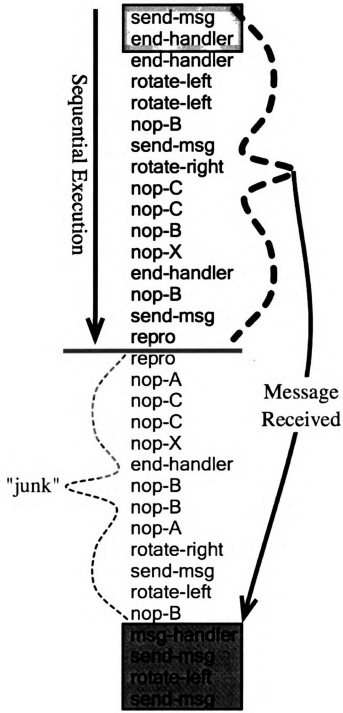


Figure 5.8: Evolved dominate genome that produces the lowest average organism density.

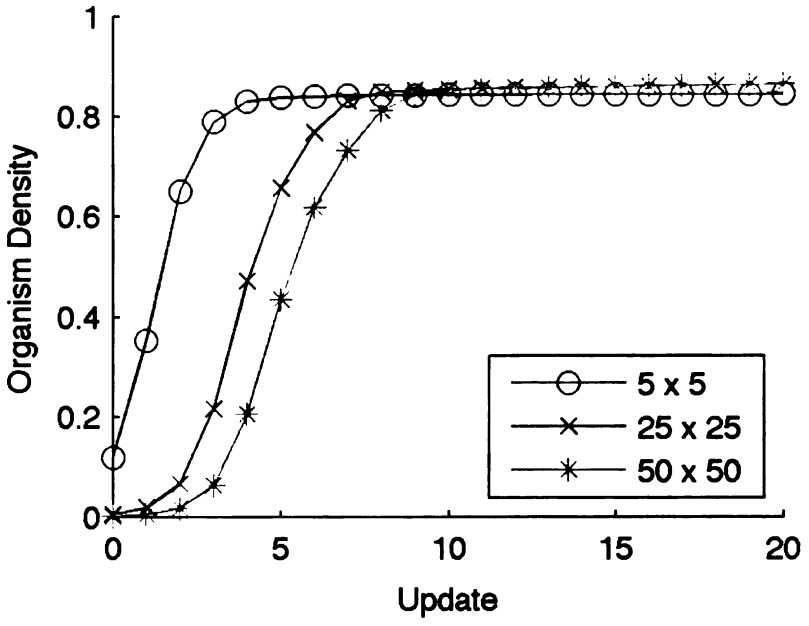


Figure 5.9: Average organism density under multiple deme sizes.

contain an organism, it is colored white. A cell that contains an uninterrupted organism is colored black, and a red cell denotes the presence of an organism in an interrupted state. As depicted in Figure 5.10, the population quickly switches behaviors when it becomes dense. In fact, the time difference between Figures 5.10(c) and 5.10(e) is less than two updates. The rapid change is behavior is a hallmark of QS and has been observed in natural and now digital organisms.

5.3 Quorum Quenching in Digital Organisms

Methods to prevent or disrupt QS (referred to as *quorum quenching*) can serve multiple purposes, from treating a disease to disabling a distributed attack. In addition, quorum quenching can be used to test the robustness of a system, revealing weaknesses in the design before the system is deployed. Indeed, quorum quenching techniques have been proposed as a means to reduce the virulence of pathogenic bacteria [163, 196]. Two major categories of treatments have emerged: those that affect AI transmission through the

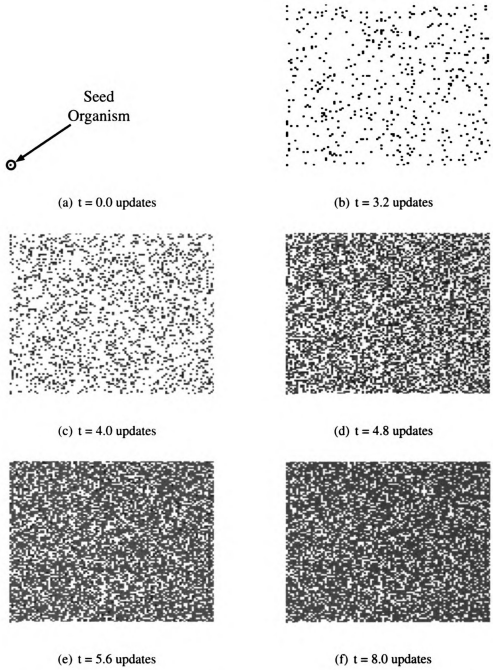


Figure 5.10: Images of a 100×100 deme initially seeded with the genome shown in Figure 5.8. Figure 5.10(a) shows the initial state of the deme with a single uninterrupted organism in the lowest, left-most cell. Figures 5.10(b) and 5.10(c) show a steady exponential increase in population size where the majority of organisms are executing sequentially. Figures 5.10(d) and 5.10(e) depict the rapid behavioral change from self-replication to suppression of self-replication. Lastly, Figure 5.10(f) shows the state of the deme population 40% of the way through a competition period.

extracellular medium [196], and those that introduce mutant bacteria incapable of sending (*signal-negative*) or receiving (*signal-blind*) AIs [163]. In this paper we investigate the evolution of resistance to these types of mutants, in an attempt to help predict outcomes and discover treatments in both biological and computational domains.

5.3.1 Experimental Setup

The investigation of medical treatments that quench QS bacteria by introducing disabled mutants has shown promise [163]. In [163], mice were infected with multiple strains of *Pseudomonas aeruginosa* and their survival rates were tracked. It was found that infections containing mutants were statistically worse at killing mice than the control. Effectively, the mutants act as *cheaters*, exploiting the cooperative production of virulence factors, but not fully participating in the underlying QS behavior. Moreover, it was shown receive-impaired mutants were more effective at reducing mortality rate than send-impaired mutants, suggesting different impairments disrupt QS activity to varying degrees.

In our Avida experiments, we explore how digital organisms fare in environments where offspring are probabilistically impaired at birth so they cannot send or receive messages. We present three treatments designed to study the evolution of QS behavior and the resistance to mutants. First, we evolve organisms in environments with constant rates of impairment. We quantify the observed differences between demes that are subjected to both types of mutants over a range of impairment introduction rates. This experiment enables us to test for environmental barriers that inhibit the evolution of QS. Next, we take evolved genomes that perform QS and subject them to environments with constant rates of impairment. We show that these evolved genomes exhibit a robustness to mutants at higher introduction rates than genomes that were exposed to constant introduction of mutants during evolution. Finally, we subject the same genomes to environments where an increasing percentage of births result in mutants, and report on the adaptation of these genomes in this environment.

In all experiments, a population is divided into 400 demes. Each deme has an identical 5×5 toroidal topology and is seeded with an organism at the beginning of each competition period. Competition periods last for 20 *updates*; an update is a unit of time in Avida equivalent to an average execution of 50 instructions per organisms. At the end of a competition period each deme's fitness is evaluated using Equation 5.1. The genomes used for the initial seed organisms vary by treatment, and will be described below as needed. All data reported in this chapter were produced using revision 3204 of the Avida source code, publicly available in the subversion repository located at `https://avida.devosoft.org/svn/branches/interrupt`.

5.3.2 Results

Evolving Resistant Organisms - Naive Method. Initially, we attempt to evolve organisms that are resistant to mutants. In this treatment, mutants are introduced at the rates of 0, 1, 2, 5, or 10 percent of all births. The creation of the mutants that cannot send a message is done by disabling the send-msg instruction. Mutants that cannot receive a message are created by disabling message receive interrupt handlers, causing all messages to remain buffered. Once a mutant is introduced, it remains a mutant for its lifetime, and its impaired abilities are inherited by all its descendants. In this treatment, a default ancestral organism, containing 49 nop instructions followed by a single repro, seeds each of 20 runs for each of the 25 mutant rate pairs. Each of these runs is seeded with a different random number seed and is allowed to evolve for 5,000 deme generations (competition periods).

Figure 5.11 plots the fraction of runs for each mutant rate pair that evolved QS behaviors. From Figure 5.11, we observe an environmental barrier to the evolution of QS. Specifically, as the percent of receive-disabled mutants increases, the number of runs that evolve QS behavior goes to zero. To evolve QS, the number of organisms that cannot receive a message must remain small, less than 5%. This result suggests that an environment with a high concentration of mutants incapable of receiving messages does not provide the

evolutionary process with enough stable building blocks by which QS can evolve, producing a barrier to the evolution of QS. Also, that the introduction of receive-impaired mutants is more effective at preventing QS than send-impaired mutants, and it therefore a more desirable treatment. Next, we test the resistance of organisms that have *already* evolved to perform QS, to the introduction of mutants.

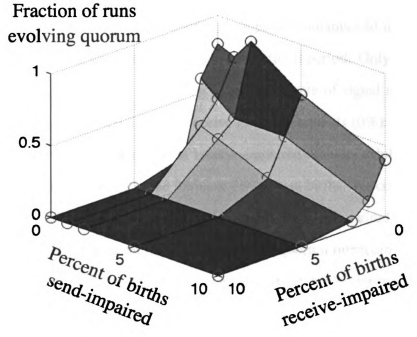


Figure 5.11: Fraction of runs that evolved a quorum mechanism under constant mutant introduction rates. Results are the mean of 20 runs sampled at each of the 25 configurations, marked by a circle.

Resistance in Quorum Sensing Organisms. In this treatment, we extended the 20 runs from [17] from 2,500 deme generations to 5,000, further optimizing the evolved genomes, and observed that 16 runs evolved QS behavior. We then tested the robustness of the 16 dominant (most abundant that the end of each run) genomes to the introduction of mutants. Using the same mutant introduction rates as in Section 5.3.2, we subjected each of these 16 dominant genomes to all 25 mutant introduction rate pairs. To perform tests similar to those done with mice in [163], we injected a single seed organism containing one of the

16 dominant genomes, into all 400 demes and measured the deme survival rate, that is, the fraction of demes that contain "living" organisms after one competition period.

Figure 5.12 shows the mean fraction of demes surviving at the conclusion of a competition period, interpolated over the range of 0 to 10 mutants per 100 births for both mutant types. As expected, when mutants are not introduced (the conditions in [17]), all of the demes contain living organisms at the conclusion of the experiment, denoted by point *A* in Figure 5.12. Moreover, the introduction of send-impaired mutants had minimal effect on the number of demes that survived for a single competition period. Only 549 out of 6400 demes were unable to survive at the highest introduction rate of signal impaired mutants (10%) and the lowest introduction rate of receive impaired mutants (0%). However, demes that were subjected to the introduction of receive-impaired mutants exhibited an increase in deme mortality corresponding to the frequency of mutant births. We conclude that these evolved genomes are more susceptible to disruption by receive-impaired mutants than by send-impaired mutants. Based on these data, a logical quorum quenching treatment is to introduce receive-impaired mutants into the demes. At a introduction rate of 10%, such mutants killed approximately 88.4% of all demes tested, or 28,297 out of 32,000.

We observe that while the dependency trends in Figure 5.12 are similar to those in Figure 5.11, a higher resistance to receive-impaired mutants in exhibited. Specifically, a 10% introduction rate does not kill all demes. The presence of this resistance is intriguing, since the dominant genomes were evolved under conditions free of explicitly introduced mutants, limiting selective pressures that may drive a population toward a resistant solution. However, we note that a pressure to build up resistance to mutants is supplied by the deme-level fitness function, which rewards demes that minimize their energy usage. Specifically, genetic mutations regularly produce degenerate variants of this behavior during the evolutionary process, providing a pressure to increase resistance to these types of mutants. This pressure produces a more robust algorithm, regardless of whether the misbehaving nodes are produced through genetic mutation or are artificially impaired and placed in the pop-

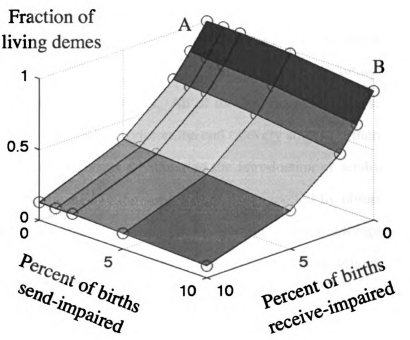


Figure 5.12: Fraction of living demes exposed to mutants after a competition period. Results are the mean of 16 runs, one for each dominant exhibiting QS, sampled at each of the 25 configurations, marked by a circle.

ulation. Extending this line of exploration, we next attempt to evolve even more resistant organisms, using an environment with a staged increase in the introduction of impaired mutants.

Evolving Resistant Organisms - Staged Method. We next focus on evolving organisms that are resistant to receive- and send-impaired mutants. We again use the QS performing dominant genomes from [17] as seed organisms. However, in this treatment the organisms are subjected to a staged increase in the level of receive- or send-impaired mutants. Specifically, during a run the impaired mutant introduction rate is increased by 1% every 500 competition periods, initially starting at 1%. We use the same experimental configurations as in the previous experiments along with the staged environment, and use each of the 16 dominants to seed 10 replicate runs.

The two previous experiments suggest that send-impaired mutants will have a smaller effect on a deme's ability to perform QS than receive-impaired mutants. This result is confirmed by Figure 5.13, which displays the fraction of runs exhibiting QS at the end of all 10 mutant introduction rate stages. All of the 160 runs that were subjected to send-impaired mutants exhibit QS behavior at the end of every stage. Furthermore, from these data it is evident that a stronger resistance to the introduction of send-impaired mutants has evolved when compared to the seed organisms, supported by observing that point *B* in Figure 5.12 shows a slight increase in deme mortality when send-impaired mutants are introduced at a rate of 10%. In contrast, none of the dominant genomes that evolve in the staged environment exhibit deme mortality at a send-impaired mutant introduction rate of 10%. Apparently, as the environment became increasingly adverse, with respect to the introduction of send-impaired mutants, the evolutionary process produced organisms that are more robust to these mutants.

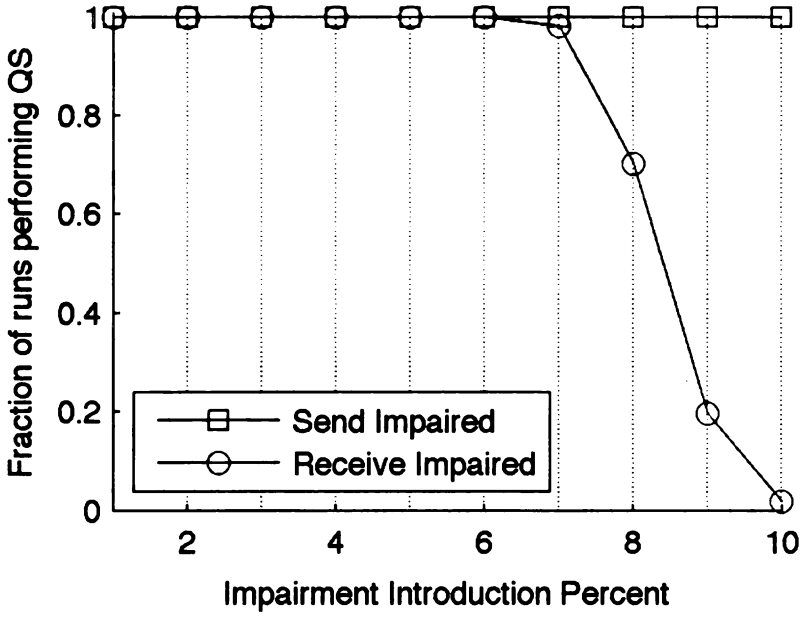


Figure 5.13: Fraction of runs subjected to impaired mutants that perform QS at the end of each stage. Results are a composite of 160 runs for each.

As expected by the trends observed in both Figures 5.11 and 5.12, receive-impaired mutants have a greater effect on the evolution of QS. This observation is reinforced in Figure 5.13, which demonstrates a rapid decline in the fraction of populations performing QS in the last three stages, concluding with only 6 of the 160 runs performing QS in the last stage. However, increased robustness is present in these evolved genomes when compared to the seed genomes. Specifically, all of the 160 dominants produced by the runs depicted in Figure 5.13 are immune to receive-impaired mutant introduction at rates from 1% to 6%, which is an increase over the resistance of the seed genomes. Moreover, an immunity to receive-impaired mutants is also present in some dominant genomes at a mutant introduction rate as high as 10%, which provides evidence that these mutants can be overcome and QS can persist. However, these data alone does not reveal whether QS is suppressed by the introduction of receive-impaired mutants or if the code for the behavior has evolved away.

To answer this question, let us explore the behavior of individual organisms. The strategy evolved in all initial QS dominant seed organisms is to repeatedly interrupt their neighbors, preventing self-replication, at quorum. By causing interrupts, an organism can effectively extend its neighbor's gestation time, assuming replication is not done within an interrupt handler. Figure 5.14 shows the mean number of organisms interrupted per deme during evolution in the staged environment. The error bars represent one standard deviation from the mean. Figure 5.14 displays a larger decline in the treatment where receive-impaired mutants are introduced than when send-impaired mutants are introduced. These data suggest that the genomes subjected to send-impaired mutants do not lose the genetic code required to handle an interrupt. However, the decline to near zero in the receive-impaired treatment indicates that either the mutants are disrupting QS to the extent that organisms are rarely interrupted, or the genetic code to become interrupted has evolved away.

In the experiment whose data are depicted in Figures 5.13 and 5.14 we observed that additional evolution in the staged environment produced organisms that are more resistant

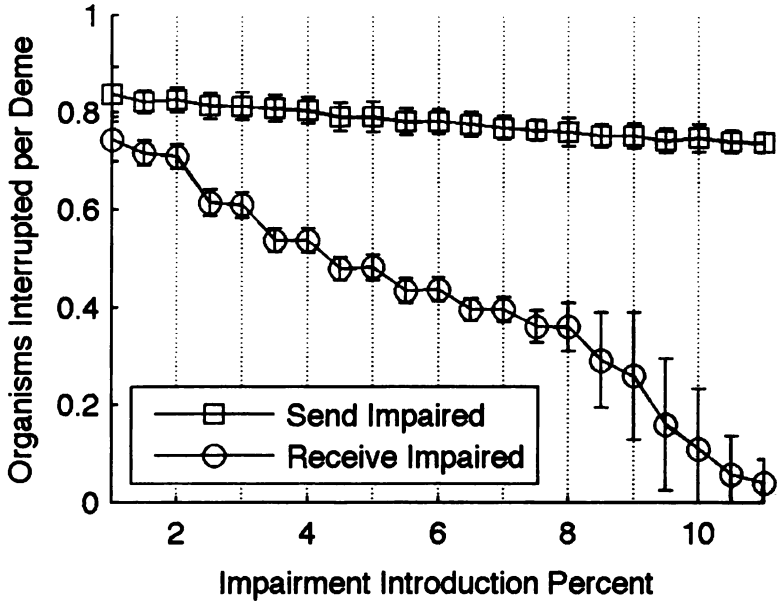


Figure 5.14: Fraction of organism interrupted per deme during evolution in the staged environment. Error bars denote one standard deviation from mean. Data are a composite of 160 runs.

to both types of mutants. To quantify the level of resistance over time we extracted dominant genomes at time points 0, 25, 50, 75, and 100 percent through each of the staged environment runs. We then tested these dominant genomes for QS behavior in the absence of mutants. These tests should reveal changes in the genomes that enhance or diminish QS. To perform these tests each dominant genome was used to create an organism which is then injected into 400 demes. The demes were allowed to execute for one competition period, and we tracked the mean organism density (fraction of organisms per deme) of all demes. We then identified the genomes that were performing QS and plotted the mean organism density of all demes for all of these dominant genomes over the course of a single competition period.

Figure 5.15 shows the mean organism density of the dominant genomes that were subjected to send-impaired mutants. These data show that as the percentage of send-imparied mutants increased, the genomes evolved to sense a quorum at a lower organism density, concluding in a mean quorum triggering density of approximately 0.65. In addition, all

pairwise comparisons of organism density at the end of a competition period, excluding the 0% and 25% pair, are significantly different according to the Wilcoxon rank sum test for equal medians using an $\alpha = 0.01$. The significant decline in organism density required to perform a quorum illustrates the effect of additional evolution with the introduction of send-impaired mutants. Specifically, a quorum is sensed, triggering behavioral change, with fewer organisms, thereby producing a more fit deme.

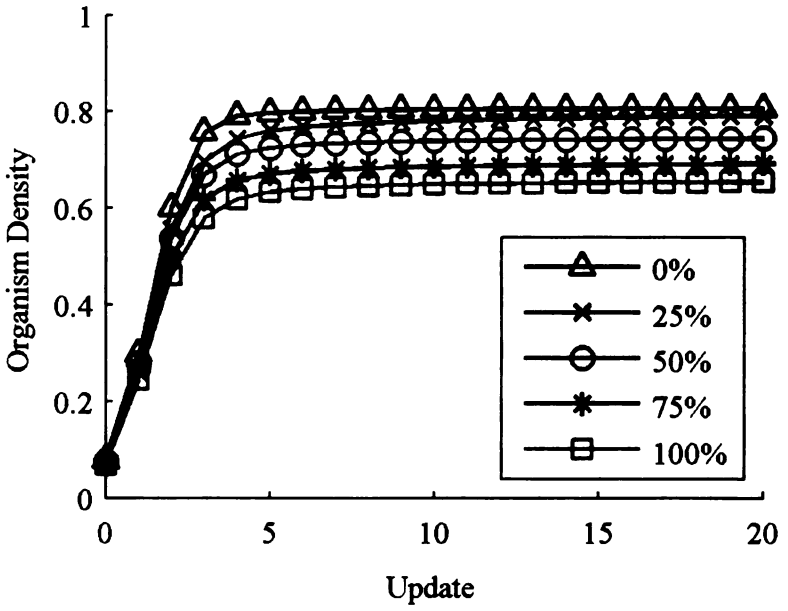


Figure 5.15: Mean organism density of dominant genomes extracted from runs exposed to a staged increase in send-impaired mutants. Error bars are omitted for clarity. Data are a composite of 160 samples per line.

Figure 5.16 shows that receive-impaired mutants also have a similar effect, reducing the number of organisms required to achieve a quorum. However, unlike send-impaired mutants, as the introduction of receive-impaired mutants increases, the time required to achieve a quorum also increases, which has a negative effect on the ability of a deme to sense a quorum. Specifically, only 6 of the 160 dominant genomes present at the end of the staged environment runs perform QS in the absence of mutants. In fact, these are the same 6 runs that perform QS in the presence of receive-impaired mutants introduced at a rate of 10%. This fact, in conjunction with the data presented in Figures 5.13 and 5.14,

demonstrates that receive-impaired mutants introduced at a rate of 10% of all births is effective at disabling QS. Furthermore, their presence causes QS behavior to evolve away in more than 96% of the runs.

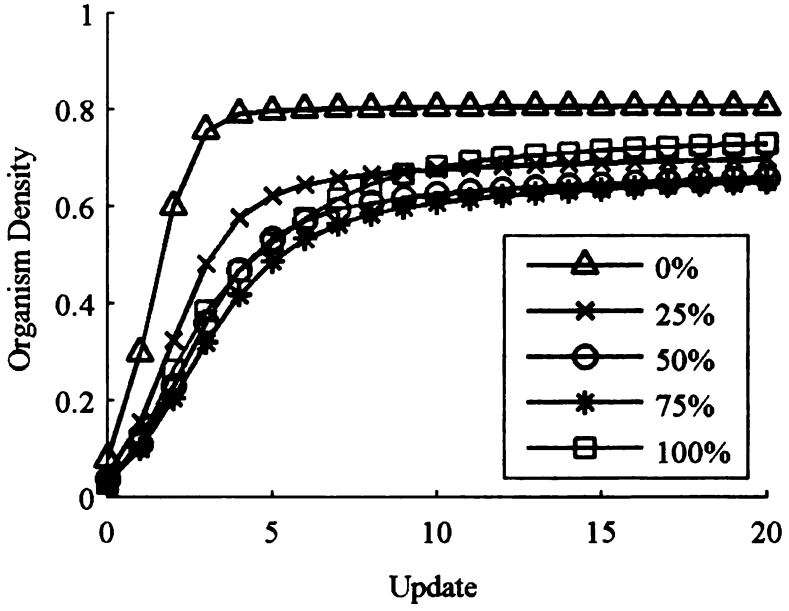


Figure 5.16: Mean organism density of dominant genomes extracted from runs exposed to a staged increase in receive-impaired mutants. Error bars are omitted for clarity.

5.4 Conclusion and Future Work

Many natural and artificial systems utilize QS to perform critical tasks, from self-preservation in bacteria to clustering in sensor networks. In addition, the disruption of these systems can serve many purposes, from treating disease-causing bacteria to disabling cyber-attacks. In this chapter we have demonstrated the evolution of QS behavior in digital organisms and responses to QQ techniques.

We examined the evolution of behavior that suppresses individual self-replication under high density conditions. First, we showed that this type of behavior evolves in a majority of runs in all three of our rotation treatments. Second, we extracted the dominant genomes

from the single rotation treatments and showed the tight correlation between organism density, total births, and number of interrupted organisms. Third, we discussed the individual behavior of the dominant genome that produced the lowest organism density. Fourth, we increased the size of the demes by two orders of magnitude and showed the average organism density remained the same. Lastly, we visualized the rapid change in behavior of a dominant organism in a deme 400 times larger than the environment it evolved in.

We also showed that digital evolution can produce solutions that are resistant to communication impairments. Furthermore, we have shown experimentally that a staged environment, where impairments are increasing introduced over time, can improve the robustness of evolved solutions. We demonstrated that QS digital organisms that have not been exposed to explicitly introduced impairments exhibit a resistance to these mutants. However, their resistance is lower than organisms exposed to a staged introduction of mutants. Finally, we showed that the introduction of receive-impaired mutants in a staged environment can cause QS behavior to evolve away.

Chapter 6

Conclusion

This dissertation provides experimental evidence to support the thesis that the incorporation of an energy producing metabolism, used for individual function, provides an additional feature by which natural selection can act to produce efficient behaviors in artificial systems. The incorporation of the energy model into Avida allows for experimental configurations where digital organisms are required to pay both time and energy costs to execute. Modeling energy costs, in addition to time costs, provides a baseline from which cooperative, energy-conserving behavior can evolve.

We have demonstrated that the inclusion of energy can fundamentally alter the trajectory of an evolving population. Compared to trials where organisms were not required to pay energy costs, experimental data demonstrated that similar levels of reaction completion and ecotype diversity are observed. However, under conditions where organisms do not pay energy costs, experimental data demonstrated that the population's size is not limited by resource inflow and individual gestation can be hard to predict, which is not true when organisms are required to pay energy costs. The inclusion of the energy model provides a basis upon which energy conserving behaviors can be evolved.

Individual energy conserving behaviors have been evolved using the energy model. Initially, resource-aware adaptive sleep/wake behavior was evolved in an environment where

resource availability is periodic and declines over time. In these experiments, organisms evolved to become highly active when the resource is available and sleep when it is not. They evolved a circadian rhythm that allowed them to awaken before the resource became available, complete and be rewarded for performing tasks while the resource was available, and then return to a sleep state prior to the point in time when the resource became unavailable. In other words, the organisms evolved an "early to bed, early to rise" strategy for consuming resources that prevented them form performing a task and receiving no reward.

In addition to evolving individual, adaptive, resource-aware behavior, we also evolved taxis. In these experiments digital organism evolved to move toward the highest concentration within a virtual, unimodal, chemical gradient. The evolved genomes that were successful at guiding an organism toward higher levels of virtual chemicals were cross compiled and loaded onto a physical robot. Once initialized, robot autonomously performed phototaxis.

The successful evolution of individual energy conserving behaviors led to multiple experiments on evolving energy-conserving group behaviors. These experiments targeted the evolution of behaviors that conserve a group's collective energy. By allowing conserved energy to be passed from generation to generation, we evolved groups of organisms that foraged for and collected information from their environment while limiting the size of their group. Specifically, we showed that a population can evolve to self-regulate its size when a small amount of the parent's collective energy is transferred to the offspring. These experiments illustrate that increases in organism gestation time conserves the groups collective energy. However, this is in contrast to the pressure to complete the group-level goal. Therefore, the foraging and collection behavior evolved to manage this tradeoff.

In another set of experiments, we showed that a group of digital organisms can evolve to adaptively control the group's size in response to environmental pressures. Specifically, to promote adaptive behavior we evolved groups of organisms in an environment containing fluctuating levels of attacks. Under these conditions the organisms evolved to conserve en-

ergy by sleeping, which helped maintain a small population, when attacks were not present in the environment. Moreover, when the attack level rose, the organisms transmitted signals that altered the execution of the recipient. The recipient's altered execution caused it to propagate the signal, attempt to quell nearby attacks, and then quickly replicate. This altered behavior managed the threat until it subsided. These experiments demonstrated that digital organisms can evolve adaptive, group-level, energy-conserving behavior.

Building on these experiments, we designed experiments to evolve a density-dependent adaptive behavior, or QS. Using a simple fitness function, where a group is evaluated based on the amount of energy it conserved, we were able to evolve organisms that replicated until a quorum was reached. Upon reaching an evolved density threshold, the organisms suppressed replication by continually interrupting the execution of other organisms. This was done by sending messages that the recipient must handle if it contains an (evolved) interrupt handler. In these experiments the organisms evolved to suppress replication at a density of 0.8 in worlds of various sizes. Additionally, the execution of their evolved interrupt handlers produced positive feedback that approximately tripled the number of signals sent within the group. Both of these behaviors mirror behaviors exhibited in bacteria.

Lastly, we performed experiments to judge the level of resistance in a QS population to messaging impairments. We showed that an evolved resistance can arise in populations that are subjected to impairments throughout evolutionary time. We also demonstrated that organisms which have not been explicitly exposed to these impairments exhibited resistance. We showed that a staged increase in the presence of impairments increased the level of resistance exhibited by the organisms. This work illustrates that a resistance to disruptive techniques can exist even when the methods of disruption are not explicitly placed in the system. Additionally, the ability of the organisms to evolve methods that mitigate the disruptions serves as evidence that anti-infective therapies many experience similar resistance in the future.

Furthering our understanding about how groups of simple agents interact to self-

organize and perform collective behaviors has benefits that span many disciplines. For example, as described in Chapter 5, bacteria perform quorum sensing to collectively make a decision. This decision process arises out of indirect communication between individuals and its disruption can alter the group's behavior. Studying the evolutionary process that produced self-organizing biological systems is a logical step toward a fuller understanding of these systems. Moreover, studying evolution using computers removes or lessens many constraints that are inherent to natural systems. Studying artificially evolved systems may lead to the discovery of behaviors that are not currently present, or even possible, in natural systems, possibly expanding the repertoire of self-organizing behaviors that can be realized and leveraged in artificial environments.

Traditional software development methods are being challenged by the ever-increasing complexity of today's software and hardware systems. As the cost of developing these systems grows, alternative methods that produce adequate solutions using less manpower are appealing. Evolution provides us with a method capable of designing solutions for increasingly complex environments. By drawing inspiration from natural systems and harnessing the evolutionary process which produced those systems, we hope to provide tools capable of handling the escalating complexity of distributed computing systems. Enabling software development methods that harness this power may provide a means to produce economical software solutions that exhibit the robustness, flexibility, and adaptability that abound in nature.

BIBLIOGRAPHY

[1] A. Abraham and C. Grosan. Evolving intrusion detection systems. In N. Nedjah, A. Abraham, and L. de Macedo Mourelle, editors, *Genetic Systems Programming: Theory and Experiences*, volume 13 of *Studies in Computational Intelligence*, pages 57–80. Springer, Germany, 2006.

[2] C. Adami. *Introduction to Artificial Life*. Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[3] C. Adami, C. A. Ofria, and T. C. Collier. Evolution of biological complexity. *Proceedings of the National Academy of Sciences*, 97(9):4463–4468, April 2000.

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

[5] R. J. Anthony. Emergence: A paradigm for robust and scalable distributed applications. In *Proceedings of the First International Conference on Autonomic Computing*, pages 132–139, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[6] J. Arabas, Z. Michalewicz, and J. J. Mulawka. GAVaPS - a genetic algorithm with varying population size. In *Proceedings of the International Conference on Evolutionary Computation*, pages 73–78, 1994.

[7] J. Ayers. *Neurotechnology for Biomimetic Robots*. MIT Press, Cambridge, MA, USA, 2002.

[8] J. Ayers, J. Witting, C. Wilbur, P. Zavracky, N. McGruer, and D. Massa. Biomimetic robots for shallow water mine countermeasures. In *In Proc. of the Autonomous Vehicles in Mine Countermeasures Symposium*, 2000.

[9] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):26–66, 2006.

[10] M. Bakhouya and J. Gaber. Adaptive approach for the regulation of a mobile agent population in a distributed network. In *Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing*, pages 360–366, Washington, DC, USA, 2006. IEEE Computer Society.

[11] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference*, page 13, Washington, DC, USA, 1998. IEEE Computer Society.

[12] G. Baldassarre, S. Nolfi, and P. D. Evolution of collective behaviour in a team of physically linked robots. In R. Gunther, A. Guillot, and J.-A. Meyer, editors, *Applications of Evolutionary Computing*, pages 581–592. Springer Verlag, Heidelberg, Germany, 2003.

[13] G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behaviours. *Artificial Life*, 9:255–267, 2002.

[14] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSUR)*, 25(2):171–220, 1993.

[15] J. Beal and J. Bachrach. Infrastructure for engineered emergence on sensor/actuator networks. *IEEE Intelligent Systems*, 21(2):10–19, 2006.

[16] B. E. Beckmann and P. K. McKinley. Evolution of adaptive population control in multi-agent systems. In *Proceedings of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 181–190, Oct. 2008.

[17] B. E. Beckmann and P. K. McKinley. Evolving quorum sensing in digital organisms. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 97–104, Montreal, QC, Canada, 2009.

[18] B. E. Beckmann, P. K. McKinley, D. B. Knoester, and C. Ofria. Evolution of cooperative information gathering in self-replicating digital organisms. In *Proceedings of 1st International Conference on Self-Adaptive and Self-Organizing Systems*, pages 65–76. IEEE Computer Society, July 2007.

[19] B. E. Beckmann, P. K. McKinley, and C. Ofria. Evolution of an adaptive sleep response in digital organisms. In *Advances in Artificial Life - Proceedings of the 9th European Conference on Artificial Life*, volume 4648 of *Lecture Notes in Computer Science*, pages 233–242. Springer, 2007.

[20] B. E. Beckmann, P. K. McKinley, and C. Ofria. Selection for group-level efficiency leads to self-regulation of population size. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 185–192, Atlanta, Georgia, 2008.

[21] O. Béjà, E. N. Spudich, J. L. Spudich, M. Leclerc, and E. F. DeLong. Proteorhodopsin phototrophy in the ocean. *Nature*, 411:786–789, 2001.

[22] F. Bernardini, M. Gheorghe, and N. Krasnogor. Quorum sensing P systems. *Theor. Comput. Sci.*, 371(1-2):20–33, 2007.

[23] A. Bieszczad, T. White, and B. Pagurek. Mobile agents for network management. *IEEE Communications Surveys*, 1998.

[24] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[25] R. Brooks. Artificial life and real robots. In *European Conference on Artificial Life*, pages 3–10, 1992.

[26] S. P. Brown and R. A. Johnstone. Cooperation in the dark: Signalling and collective action in quorum-sensing bacteria. *Proc. of the Royal Society of London B: Biological Sciences*, 268(1470):961–965, 2001.

[27] B. D. Bryant and R. Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 3, pages 2194–2201, 2003.

[28] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2006.

[29] R. Calinescu. Model-driven autonomic architecture. In *4th International Conference on Autonomic Computing*, page 9, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[30] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, and G. Theraula. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, USA, 2001.

[31] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Auton. Robots*, 4(1):7–27, 1997.

[32] G. D. Caro, F. Ducatelle, and L. M. Gambardella. Swarm intelligence for routing in mobile ad hoc networks. In *Proceesings of Swarm Intelligence Symposium (SIS 2005)*, pages 76– 83. IEEE, June 2005.

[33] R. R. Cazangi, F. J. V. Zuben, and M. F. Figueiredo. *Stigmergic Autonomous Navigation in Collective Robotics*, volume 31 of *Studies in Computational Intelligence*, chapter 2. Springer Berlin / Heidelberg, 2006.

[34] G. Chockler, S. Gilbert, and B. Patt-Shamir. Communication-efficient probabilistic quorum systems for sensor networks. In *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 111–117, Los Alamitos, CA, USA, March 2006. IEEE Computer Society.

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

[36] T. Cooper and C. Ofria. Evolution of stable ecosystems in populations of digital organisms. In *Eighth International Conference on Artificial Life*, pages 227–232, Sydney NSW Australlia, 2002.

[37] B. Crespi. The evolution of social behavior in microorganisms. *Trends in Ecology and Evolution*, 16(4), 2001.

[38] F. Crick and G. Mitchison. The function of dream sleep. *Nature*, 304(5922):111–114, 1983.

[39] D. B. D'Ambrosio and K. O. Stanley. Generative encoding for multiagent learning. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 819–826, New York, NY, USA, 2008. ACM.

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

[41] K. Dautenhahn. Getting to know each other – artificial social intelligence for autonomous robots. *Robotics and Autonomous Systems*, 16:333–356, 1995.

[42] D. Davies, M. Parsek, J. Pearson, B. Iglewski, J. Costerton, and E. Greenberg. The involvement of cell-to-cell signals in the development of a bacterial biofilm. *Science*, 280(5361):295–8, Apr. 1998.

[43] R. Dawkins. *The Selfish Gene*. Oxford University Press, New York, 1976.

[44] R. Dawkins. *The Extended Phenotype*. Oxford University Press, 2008.

[45] K. A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[46] T. R. de Kievit and I. B. H. Bacterial quorum sensing in pathogenic relationships. *Infect. Immun.*, 68(9):4839–4849, September 2000.

[47] T. R. de Kievit and B. H. Iglewski. Bacterial quorum sensing in pathogenic relationships. *Infect. Immun.*, 68(9):4839–4849, September 2000.

[48] J. A. G. M. de Visser, J. Hermisson, G. P. Wagner, L. A. Meyers, H. Bagheri-Chaichian, J. L. Blanchard, L. Chao, J. M. Cheverud, S. F. Elena, W. Fontana, G. Gibson, T. F. Hansen, D. Krakauer, R. C. Lewontin, C. Ofria, S. H. Rice, G. von Dassow, A. Wagner, , and M. C. Whitlock. Evolution and detection of genetic robustness. *Evolution*, 57:1959–1972, 2003.

[49] D. C. Dennett. The new replicators. In M. Pagel, editor, *The Encyclopedia of Evolution*, volume 1, pages E83–E92. Oxford University Press, 2002.

[50] R. J. Denver, N. Mirhadi, and M. Phillips. Adaptive plasticity in amphibian metamorphosis: Response of Scaphiopus Hammondiitadpoles to habitat desiccation. In *Ecology*, volume 79, pages 1859–1872. Ecological Society of America, 1998.

[51] G. Di Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In *In Proceedings of PPSN VIII - Eight International Conference on Parallel Problem Solving from Nature*, number 3242 in Lecture Notes in Computer Science, pages 461–470, Birmingham, UK, Sept. 2004. Springer-Verlag. Best paper award.

[52] M. Dorigo. Swarmanoid project. http://www.swarmanoid.org, January 2008.

[53] M. Dorigo, G. D. Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artif. Life*, 5(2):137–172, 1999.

[54] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[55] 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*, 17(2–3):223–245, 2004.

[56] M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. Halva, S. Nouyan, C. Ampatzis, J. louis Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L. M. Gambardella. The swarm-bots project. In *Kunstliche Intelligenz*, pages 31–44. Springer Verlag, 2005.

[57] F. Dressler, B. Kr"uger, G. Fuchs, and R. German. Self-organization in sensor networks using bio-inspired mechanisms. In *18th GI/ITG/ACM International Conference on Architecture of Computing Systems - System Aspects in Organic and Pervasive Computing (ARCS'05): Workshop Self-Organization and Emergence*, pages 139–144, mar 2005.

[58] W. Du, L. Fang, and N. Peng. LAD: localization anomaly detection for wireless sensor networks. *J. Parallel Distrib. Comput.*, 66(7):874–886, 2006.

[59] M. Dworkin and D. Kaiser. Cell interactions in myxobacterial growth and development. *Science*, 230(4721):18–24, Oct. 1985.

[60] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, March 2001.

[61] J. M. Epstein and R. Axtell. *Growing Artificial Societies: Social Science From the Bottom Up*. Brookings Institution Press and MIT Press, 1996.

[62] H. Eskandari, C. D. Geiger, and G. B. Lamont. FastPGA: A dynamic population sizing approach for solving expensive multiobjective optimization problems. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2006.

[63] C. Fernandes and A. C. Rosa. Self-regulated population size in evolutionary algorithms. In *PPSN*, volume 4193 of *Lecture Notes in Computer Science*, pages 920–929. Springer, 2006.

[64] D. Floreano, S. Mitri, S. Magnenat, and L. Keller. Evolutionary conditions for the emergence of communication in robots. *Current Biology*, 17:514–519, March 2007.

[65] D. Floreano and F. Mondada. Evolution of Homing Navigation in a Real Mobile Robot. *IEEE Transactions on Systems, Man and Cybernetics Part B : Cybernetics*, 26(3):396–407, 1996.

[66] C.-L. Fok, G.-C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *IPSN*, pages 382–387. ACM SIGBED and IEEE Signal Processing Society, IEEE, April 2005.

[67] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 653–662, Washington, DC, USA, 2005. IEEE Computer Society.

[68] S. Forrest and S. Hofmeyr. Engineering an immune system. *Graft*, 4(5):5–9, 2001.

[69] M. Gardner. Mathematical games the fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223:120–123, October 1970.

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

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

[72] H. J. Goldsby, D. B. Knoester, B. H. C. Cheng, P. K. McKinley, and C. A. Ofria. Digitally evolving models for dynamically adaptive systems. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Minneapolis, Minnesota, May 2007.

[73] F. Gomez and R. Mikkulainen. Incremental evolution of complex general behavior. *Adapt. Behav.*, 5(3-4):317–342, 1997.

[74] R. Groß and M. Dorigo. Cooperative transport of objects of different shapes and sizes. In *Proceedings of Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 107–118. Springer Verlag, 2004.

[75] R. Groß and M. Dorigo. Evolving a cooperative transport behavior for two simple robots. In *EA 2003*, Lecture Notes in Computer Science, pages 305–317, Berlin, Germany, 2004. Springer Verlag.

[76] R. Groß and M. Dorigo. Group transport of an object to a target that only some group members may sense. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – 8th International Conference (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, pages 852–861. Springer Verlag, Berlin, Germany, 2004.

[77] M. Guimarães and L. Rodrigues. A genetic algorithm for multicast mapping in publish-subscribe systems. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications*, Washington, DC, USA, 2003. IEEE Computer Society.

[78] S. Hansen, P. Rainey, J. Haagensen, and S. Molin. Evolution of species interactions in a biofilm community. *Nature*, 445(7127):533–536, Feb 2007.

[79] S. K. Hansen, P. B. Rainey, J. A. J. Haagensen, and S. Molin. Evolution of species interactions in a biofilm community. *Nature*, 445:533–536, February 2007.

[80] A. Haque. Psychology from islamic perspective: Contributions of early muslim scholars and challenges to contemporary muslim psychologists. *Journal of Religion and Health*, 43(4):357–377, 2004.

[81] I. Harvey, E. D. Paolo, R. Wood, M. Quinn, and E. Tuci. Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11(1-2):79–98, 2005.

[82] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.

[83] S. A. Hofmeyr and S. A. Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473, 2000.

[84] 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.

[85] J. H. Holland. *Hidden order: How adaptation builds complexity.* Helix Books, 1995.

[86] J. H. Holland. Echoing emergence: Objectives, rough definitions, and speculations for echo–class models. *Complexity: Metaphors, Models and Reality*, 1996.

[87] B. Holldobler and E. O. Wilson. *The Ants.* Springer, Berlin, 1990.

[88] A. M. Hoover and R. S. Fearing. Fast scale prototyping for folded millirobots. *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1777–1778, May 2008.

[89] G. S. Hornby, H. Lipson, and J. B. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[90] P. Hraber, T. Jones, and S. Forrest. The ecology of echo. *Artificial Life*, 3(3):165–190, 1997.

[91] W. Huang, C. Ofria, and E. Torng. Measuring biological complexity in digital organisms. In *Ninth International Conference on Artificial Life*, pages 315–321, Boston MA, 2004.

[92] M. M. Humphries, D. W. Thomas, and D. L. Kramer. The role of energy availability in mammalian hibernation: A cost-benefit approach. In *Physiological and Biochemical Zoology*, volume 76, pages 165–179. University of Chicago Press, March-April 2003.

[93] P. Husbands and J. A. Meyer. *Evolutionary Robotics*. Springer Verlag, 1998.

[94] IBM. Autonomic computing: IBM's perspective on the state of information technology. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[95] R. Jeanne. Regulation of nest construction behavior in polybia occidentalis. *Animal Behavior*, 52:473–488, 1996.

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

[97] G. H. Johnsen and P. J. Jakobsen. The effect of food limitation on vertical migration in Daphnia Longispina. In *Limnology and Oceanography*, volume 32, pages 873–880. American Society of Limnology and Oceanography, 1987.

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

[99] D. Jung and A. Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8:269–292, 2000.

[100] J. Kennedy, R. C. Eberhart, and with Yuhui Shi. *Swarm Intelligence*. The Morgan Kaufmann series in evolutionary computation. Morgan Kaufmann Publishers, 2001.

[101] P. J. Kennedy and T. R. Osborn. Operon expression and regulation with spiders. In D. Whitley, D. Goldberg, and E. Cantu-Paz, editors, *2000 Genetic and Evolutionary Computation Conf. Workshop Program*, pages 161–166, 2000.

[102] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[103] C. Killian. *Modern Control Technology*. Thomson Delmar Learning, 2005.

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

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

[106] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Mass., 1992.

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

[108] E. Kubinyi, A. Miklosi, F. Kaplan, M. Gacsi, J. Topal, and V. Csanyi. Social behaviour of dogs encoutering aibo, an animal-like robot in a neutral and in a feeding situation. *Behavioural processes*, 65:231–239, 2004.

[109] K. J. Kwak, Y. M. Baryshnikov, and E. G. Coffman. Self-organizing sleep-wake sensor systems. *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pages 393–402, Oct. 2008.

[110] T. H. Labella, M. Dorigo, and J. L. Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25, 2006.

[111] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Self-organised task allocation in a swarm of robots. In *Distributed Autonomous Robotic Systems 6, Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems*, pages 389–398. Springer-Verlag, Tokyo, Japan, 2006. In press.

[112] T. H. Labella, M. Dorigo, and J. louis Deneubourg. Efficiency and task allocation in prey retrieval. In *Proceedings of the First International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT2004), Lecture Notes in Computer Science*, pages 32–47. Springer Verlag, 2004.

[113] C. G. Langton. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120–149, 1986.

[114] C. G. Langton. *Artificial Life: An Overview*. MIT Press, Cambridge, MA, USA, 1995.

[115] R. E. Lenki, C. A. Ofria, T. C. Collier, and C. Adami. Genome complexity, robustness and genetic interactions in digital organisms. *Nature*, 400:661–664, 1999.

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

[117] H. Lipson. Uncontrolled engineering: A review of evolutionary robotics. *Artificial Life*, 7(4):419–424, 2001. book review.

[118] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, August 2000.

[119] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[120] F. Machida, M. Kawato, and Y. Maeno. Just-in-time server provisioning using virtual machine standby and request prediction. *icac*, 0:163–171, 2008.

[121] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

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

[123] D. Marocco and S. Nolfi. Emergence of communication in teams of embodied and situated agents. In *Proceedings of the 6th Evolution of Language Conference*, 2006.

[124] R. C. Massey, M. J. Horsburgh, G. Lina, M. Höök, , and M. Recker. The evolution and maintenance of virulence in staphylococcus aureus: a role for host-to-host transmission? *Nature Reviews Microbiology*, 4:953–958, December 2006.

[125] G. McHale and P. Husbands. Incorporating energy expenditure into evolutionary robotics fitness measures. In *Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 206 – 212, Cambridge, MA, USA, 2006. MIT Press.

[126] P. McKinley, B. Cheng, C. Ofria, D. Knoester, B. Beckmann, and H. Goldsby. Harnessing digital evolution. *IEEE Computer*, 41(1):54–63, January 2008.

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

[128] P. K. Mckinley, R. E. K. Stirewalt, B. H. C. Cheng, L. K. Dillon, and E. Kulkarni. Rapidware: Component-based development of adaptive and dependable middleware (www.cse.msu.edu/rapidware), 2005.

[129] N. A. Melchior and W. D. Smart. Autonomic systems for mobile robots. *icac*, 00:280–281, 2004.

[130] F. Menczer and R. K. Belew. *Latent energy environments*, pages 191–208. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[131] R. Miikkulainen and K. O. Stanley. Evolving neural networks. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 2829–2848, New York, NY, USA, 2008. ACM.

[132] M. B. Miller and B. L. Bassler. Quorum sensing in bacteria. *Annu. Rev. Microbiol*, 55:165–199, 2001.

[133] M. Mirolli and D. Parisi. Artificial organisms that sleep. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 377–386, Dortmund, Germany, September 2003. Springer.

[134] D. Misevic, R. E. Lenski, and C. Ofria. Sexual reproduction and muller's ratchet in digital organisms. In *Ninth International Conference on Artificial Life*, pages 340–345, Boston MA, 2004.

[135] D. Misevic, C. Ofria, and R. E. Lenski. Sexual reproduction reshapes the genetic architecture of digital organisms. *Proceedings of the Royal Society of London B*, 2005.

[136] W. H. Moorcroft. *Sleep, Dreaming, and Sleep Disorders*. University Press of America, Inc., MD, 2nd edition, 1993.

[137] D. E. Moriarty. *Symbiotic evolution of neural networks in sequential decision tasks*. PhD thesis, Austin, TX, USA, 1998.

[138] D. E. Moriarty, R. Miikkulainen, and P. Kaelbling. Efficient reinforcement learning through symbiotic evolution. In *Machine Learning*, pages 11–32, 1996.

[139] B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *Network, IEEE*, 8(3):26–41, May/Jun 1994.

[140] C. D. Nadell, J. B. Xavier, S. A. Levin, and K. R. Foster. The evolution of quorum sensing in bacterial biofilms. *PLoS Biology*, 6(1):171–179, January 2008.

[141] C. D. Nadell, J. B. Xavier, S. A. Levin, and K. R. Foster. The evolution of quorum sensing in bacterial biofilms. *PLoS Biology*, 6(1):0171–0179, January 2008.

[142] K. Nealson and J. Hastings. Bacterial bioluminescence: Its control and ecological significance. *Microbiol. Rev.*, 43(4):496–518, 1979.

[143] K. H. Nealson, T. Platt, and J. W. Hastings. Cellular control of the synthesis and activity of the bacterial luminescent system. *J Bacteriol.*, 104(1):313–322, 1970.

[144] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, Cambridge, MA, 2001. 2001 (2nd print), 2000 (1st print).

[145] C. Ofria and C. Adami. Evolution of genetic organization in digital organisms. In *Proc. of DIMACS workshop Evolution as Computation*, pages 167–175, Princeton, NJ, 1999.

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

[147] C. Ofria, C. Adami, and T. C. Collier. Selective pressures on genomes in molecular evolution. *J. Theor. Biology*, 222:477–483, 2003.

[148] C. Ofria, C. Adami, T. C. Collier, and G. K. Hsu. The evolution of differentiated expression patterns in digital organisms. *Lect. Notes Artif. Intell.*, 1674:129–138, 1999.

[149] C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Artificial Life*, 10:191–229, March 2004.

[150] C. Ofria and C. O. Wilke. Avida: Evolution experiments with self-replicating computer programs. In A. Adamatzky and M. Komosinski, editors, *Artificial Life Models in Software*, pages 3–35. Springer Verlag, London, 2005.

[151] A. N. Pargellis. Digital life behavior in the amoeba world. *Artificial Life*, 7(1):63–75, 2000.

[152] J. Park, R. Jagasia, G. F. Kaufmann, J. C. Mathison, D. I. Ruiz, J. A. Moss, M. M. Meijler, R. J. Ulevitch, and K. D. Janda. Infection control by antibody disruption of bacterial quorum sensing signaling. *Chemistry & Biology*, 14(10):1119–1127, Oct. 2007.

[153] S. Pierre and G. Legault. A genetic algorithm for designing distributed computer network topologies. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 28(2):249–258, Apr 1998.

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

[155] J. R. Porter. Antony van leeuwenhoek: tercentenary of his discovery of bacteria. *Microbiol. Mol. Biol. Rev.*, 40(2):260–269, 1976.

[156] J. Pugh and A. Martinoli. Multi-robot learning with particle swarm optimization. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 441–448, New York, NY, USA, 2006. ACM.

[157] S. Rasmussen, C. Knudson, P. Feldberg, and M. Hindsholm. The coreworld: Emergence and evolution of cooperative structures in a computational chemistry. *Physica D*, 42(1-3):111–134, 1990.

[158] T. S. Ray. An approach to the synthesis of life. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 371–408. Addison-Wesley, Reading, MA, USA, 1992.

[159] T. S. Ray. Evolution, ecology and optimization of digital organisms. Working Paper 92-08-042, Santa Fe Institute, 1992.

[160] T. S. Ray. Evolution, complexity, entropy and artificial reality. In *Proceedings of the Oji international seminar on Complex systems : from complex dynamical systems to sciences of artificial reality*, pages 239–263, New York, NY, USA, 1994. Elsevier North-Holland, Inc.

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

[162] F. J. Romero-Campero and M. J. Pérez-Jiménez. A model of the quorum sensing system in Vibrio fischeri using P systems. *Artificial Life*, 14(1):95–109, 2008.

[163] K. P. Rumbaugh, S. P. Diggle, C. M. Watters, A. Ross-Gillespie, A. S. Griffin, and S. A. West. Quorum sensing and the social evolution of bacterial virulence. *Current Biology*, 19(4):341–345, January 2009.

[164] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, 3:2478–2483 vol.3, 2002.

[165] K. L. Sauer, A. Camper, G. Ehrlich, J. Costerton, and D. Davies. Pseudomonas aeruginosa displays multiple phenotypes during development as a biofilm. *Journal of Bacteriology*, 184(4):1140–1154, 2002.

[166] S. Schauder, K. Shokat, M. Surette, and B. Bassler. The LuxS family of bacterial autoinducers: biosynthesis of a novel quorum-sensing signal molecul. *Mol Microbiol*, 41(2):463–476, July 2001.

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

[168] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[169] K. Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 1994. ACM.

[170] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 2009. In press.

[171] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, 2002.

[172] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Auton. Robots*, 8(3):345–383, 2000.

[173] J. Strassner. Focale - a novel autonomic networking architecture. In *First Latin American Autonomic Computing Conference (LAACS)*, July 2006.

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

[175] X. Tan, D. Kim, N. Usher, D. Laboy, J. Jackson, A. Kapetanovic, J. Rapai, B. Sabadus, and X. Zhou. An autonomous robotic fish for mobile sensing. In *Proceedings of International Conference on Intelligent Robots and Systems*, pages 5424–5429, Oct. 2006.

[176] G. Terrazas, N. Krasnogor, M. Gheorghe, F. Bernardini, S. Diggle, and M. Cámara. An environment aware P-System model of quorum sensing. In S. B. Cooper, B. Löwe, and L. Torenvliet, editors, *CiE*, volume 3526 of *Lecture Notes in Computer Science*, pages 479–485. Springer, 2005.

[177] V. Trianni, R. Groß, T. H. Labella, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In *Proceedings of the Seventh European Conference on Artificial Life, volume 2801 of Lecture Notes in Artificial Intelligence*, pages 865–874. Springer Verlag, 2003.

[178] V. Trianni, S. Nolfi, and M. Dorigo. Hole avoidance: Experiments in coordinated motion on rough terrain. In *Intelligent Autonomous Systems 8*, pages 29–36. IOS Press, 2004.

[179] V. Trianni, E. Tuci, , and M. Dorigo. Evolving functional self-assembling in a swarm of autonomous robots. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakamur, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8. Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB 04)*, pages 405–414. MIT Press, Cambridge, MA, 2004.

[180] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. Nasa's swarm missions: The challenge of building autonomous software. *IT Professional*, 6(5):47–52, 2004.

[181] E. Tuci, C. Ampatzis, F. Vicentini, and M. Dorigo. Evolved homogeneous neurocontrollers for robots with different sensory capabilities: Coordinated motion and cooperation. In S. Nolfi, G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, editors, *From Animals to Animats 9: 9th International Conference on Simulation of Adaptive Behavior, SAB 2006*, volume 4095, pages 679–690. Springer Verlag, Berlin, Germany, 2006.

[182] R. Vogt, J. Aycock, and M. J. Jacobson. Quorum sensing and self-stopping worms. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, pages 16–22, New York, NY, USA, 2007. ACM.

[183] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 256–266, New York, NY, USA, 1992. ACM.

[184] T. Wang, C. C. Hung, and D. J. Randall. The comparative physiology of food deprivation: From feast to famine. In *Annual Review of Physiology*, volume 68, pages 223–251. Annual Reviews, 2006.

[185] C. M. Waters and B. L. Bassler. Quorum sensing: Cell-to-cell communication in bacteria. *Annual Review of Cell and Developmental Biology*, 21:319–346, 2005.

[186] G. Wei. Adaptation and learning in multi-agents systems: Some remarks and a bibliography. In *Adaptation and Learning in Multi-Agent Systems*, pages 1–21. Springer-Verlag, 1995.

[187] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An architectural approach to autonomic computing. In *Proceedings of the First International Conference on Autonomic Computing*, pages 2–9, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[188] T. White, B. Pagurek, and D. Deugo. Management of mobile agent systems using social insect metaphors. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, pages 410–415, 2002.

[189] C. O. Wilke and C. Adami. The biology of digital organisms. *TRENDS in Ecology & Evolution*, pages 528–532, 2002.

[190] C. O. Wilke, J. Wang, C. A. Ofria, C. Adami, and R. E. Lenki. Evolution of digital organisms at high mutation rate leads to survival of the flattest. *Nature*, 412:331–333, 2001.

[191] D. S. Wilson. Introduction: Multilevel selection theory comes of age. *The American Naturalist*, 150(S1-S4), July 1997.

[192] E. O. Wilson. *Sociobiology: The New Synthesis*. Harvard University Press, 1975.

[193] I. Wokoma, L. Sacks, and I. W. Marshall. Clustering in sensor networks using quorum sensing. In *London Communications Symposium*, University College London, September 2003.

[194] T. D. Wolf, G. Samaey, T. Holvoet, and D. Roose. Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical methods. In *Proceedings of the Second International Conference on Autonomic Computing*, pages 52–63, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[195] R. J. Wood, S. Avadhanula, M. Menon, and R. S. Fearing. Microrobotics using composite materials: the micromechanical flying insect thorax. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 1842–1849. IEEE, September 2003.

[196] K. B. Xavier and B. L. Bassler. Interference with AI-2-mediated bacterial cell-cell communication. *Nature*, (437):750–753, 2005.

[197] C. H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. Technical report, Austin, TX, USA, 2001.

[198] R. A. Young. Fat, energy and mammalian survival. In *American Zoologist*, volume 16, pages 699–710. The Society for Integrative and Comparative Biology, 1976.

[199] H. Zepelin. Mammalian sleep. In M. Kryger, T. Roth, and W. Dement, editors, *Principles and Practice of Sleep Medicine*, pages 82–92. Saunders, Philadelphia, 2000.