This is to certify that the
dissertation entitled

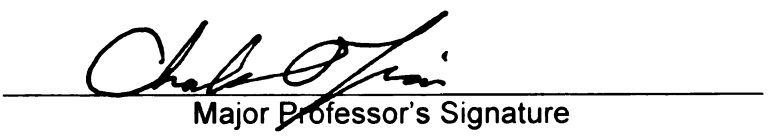
THE EVOLUTIONARY ORIGINS OF MEMORY USE IN NAVIGATION

presented by

LAURA M. GRABOWSKI

has been accepted towards fulfillment
of the requirements for the

Ph.D. degree in Computer Science



Major Professor's Signature

5-27-09

Date

THE EVOLUTIONARY ORIGINS OF MEMORY USE IN NAVIGATION

By

Laura M. Grabowski

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Computer Science

2009

ABSTRACT

THE EVOLUTIONARY ORIGINS OF MEMORY USE IN NAVIGATION

By

Laura M. Grabowski

Many organisms are able to cope with environmental variations through an array of innate behaviors. When environments change too quickly for evolutionary processes, or when the variation is more unique to a particular place, time, or individual organism's experiences, innate responses may need to be augmented by more flexible responses that use individual life experience. In such situations, memory becomes necessary. Generally, memory refers to the stored record of experience, and is of great interest across many disciplines.

I focus this study on the question of how evolution produces complex memory structures, by examining the interaction of environment and memory during evolution in an environment where navigation is an important behavior. Digital organisms evolve to navigate within their environments, based on different sensory cues. The different environments present differing problems that require different uses of memory.

The simplest environment is an idealized version of the chemical attractant gradients in the environments of bacteria such as *Escherichia coli.* These experiments demonstrated the evolution of both a chemotaxis-like response (*i.e.*, organisms evolved to move up the virtual gradient and approach the highest concentration), and a rudimentary one-timestep memory.

Inspired by maze-learning experiments with bees, in the next suite of experiments, the digital organisms evolved in environments with "paths," formed by sensory cues in the environment. I used several different types of paths, and each path type placed different demands for memory use on the evolving organisms. The results of the first group of path-following experiments demonstrated the evolution of "reflex" actions, where organisms responded in a fixed way, but differentially, to the environmental cues. For the second suite of experiments, I focused on evolving a

one-bit individual memory. One experiment evolved a one-bit life-long memory, *i.e.*, remembering a binary decision, in the form of which direction to turn in the current environment. This meant that an organism needed to store and refer to a single individual experience in future decision-making, but the content of the information did not change during the individual's lifetime. In the next set of experiments, I focused on evolving a volatile, "short-term" individual memory. In these environments, new experiences that influenced future decisions happened at unpredictable times, requiring the the stored experiences to be updated frequently.

The results of the experiments demonstrate that robust, flexible behavior can evolve even in simple environments. Organisms that evolved in each of the experimental environments exhibited clever and flexible behavior that demonstrated simple behavioral intelligence, revealing capacities such as gathering and differential use of environmental information, and the ability to use prior individual experience to guide future actions.

DEDICATION

To my family, who supported me always, and endured so much: my husband Tom Grabowski, daughters Caitlin and Miranda, my parents Pat and Lee Miesle, my sisters Julie Grossman and Paula May.

ACKNOWLEDGMENTS

I gratefully acknowledge the following people for their contributions and assistance with this work.

First, many thanks to my outstanding committee, co-advisors Dr. Charles Ofria and Dr. Robert T. Pennock, and committee members Dr. Fred. C. Dyer and Dr. Philip K. McKinley, for their mentoring, support, encouragement, and tolerance. I am deeply grateful for all that I have learned from each of you, and I feel privileged to have the benefit of your help and advice.

I owe much to my colleagues in the MSU Digital Evolution Laboratory. I would like to thank several members of the lab who made significant contributions to this work. David M. Bryson had the initial inspiration for the "dream grid" that streamlined my work in the project, and also wrote the implementation. Dr. Wesley R. Elsberry provided much assistance in many areas, from ideas, to proof-reading, to LaTeX debugging, to moral and social support. Brian Baer helped me repeatedly with all sorts of technical issues, even when I should have been able to fix them myself. Benjamin E. Beckmann and David B. Knoester helped with early experimental setups for this work, and were always ready with helpful suggestions and new ideas for improvement. Heather J. Goldsby graciously shared her presentation format with me for my defense. Dr. Chris Strelioff helped me with some pesky LaTeX problems. Thanks also to all the lab members for lots of ideas and hard questions along the way.

I am fortunate to have many supportive friends, both personal and professional. Several of them deserve special recognition for discussions and ideas as this work took shape. Drs. Katy and Dirk Luchini Colbry provided both technical and moral support, and some excellent dinners. Dr. Diane Blackwood helped with statistics, and lots of camaraderie on weekends. Pamela Rose Fox and Dr. Ann Kathleen Shea helped by always believing that I could succeed at this.

Thanks to each of you, and all those whom I neglected to include here, for giving me the tools and the fortitude to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Images in the dissertation are presented in color.

Chapter 1

Introduction and Background

1.1 Introduction

1.1.1 Motivations

More than fifty years ago, Herbert Simon and Allen Newell created the first artificial intelligence (AI) system, Logic Theorist (Newell & Simon, 1956). At the time, the outlook for AI seemed bright, and many computer scientists optimistically projected that machines with human-level intelligence were no more than a couple of decades away, at worst. More than half a century later, this optimism has dimmed considerably. Despite many impressive successes in AI and dramatic improvements in hardware, the ability to create machines with high-level, general purpose intelligence continues to seem decades away, at best.

Why has AI been unable to deliver on that early promise? Undoubtedly, there are many contributing factors. One aspect of the difficulty is that human-level intelligence is a highly complex facility, or system of facilities. In discussing Logic Theorist, Newell and Simon (1956, p. 61) identified characteristics of these types of complex systems:

> 1. There is a large number of different kinds of processes, all of which are important, although not necessarily essential, to the performance of the total system;

2. The uses of the processes are not fixed and invariable, but are highly contingent upon the outcome of previous processes and on information received from the environment;

3. The same processes are used in many different contexts to accomplish similar functions towards different ends, and this often results in organizations of processes that are hierarchical, iterative, and recursive in nature.

In many ways, this description of complexity is a good fit for both artificial and biological systems. The idea of complexity leads to another aspect of the failure of AI. The perspective that drove the first thirty to forty years of AI research was highly anthropocentric, focusing on mimicking human cognition and attempting to recreate aspects of the most complex intelligence known from scratch. This perspective placed much emphasis on higher-order propositional intelligence. *Propositional intelligence* relates to the human capacity for reasoning and abstract thought and enables humans to do things such as solve logic problems and play chess. This also overlooks the fact that, in general, humans are not very good at pure propositional logic, and tend to use a more fuzzy form. This approach tended to overlook other, simpler forms of intelligence. Many species demonstrate what is termed *behavioral intelligence*, intelligence that relates to effective, successful behavior. The interest in propositional intelligence—knowing *that*—rather than behavioral intelligence—knowing *how*—necessitated a top-down approach to designing intelligent machines. There is an obvious problem with using top-down methodology to approach intelligence: we still know relatively little about the computations and mechanisms of intelligence. Moreover, this was certainly not the way that biology produced intelligence. Evidence from evolutionary biology indicates that, just as the modern human body evolved from earlier physical forms, modern human intelligence evolved from earlier forms of intelligence. Therefore, rather than undertaking the Herculean task of designing AI from the top down, we can instead take a page from nature's book and explore intelligence from the bottom up, finding the circumstances that permit intelligence to evolve. This bottom-up approach allows us to examine the

evolution of simpler, fundamental capabilities that may facilitate the emergence of more complex intelligence.

There are many aspects to intelligent behavior. A key capability is memory. Memory is a necessary condition for many kinds of intelligent behavior. Without memory, organisms cannot draw from their own experiences to make decisions. If organisms have no memory, they cannot learn, they cannot reliably return to important places in the environment, and they cannot recognize many features of the world around them. Memory provides a tool that organisms can use to move beyond the here-and-now, using their record of experience to build predictions of future outcomes. In this study, I investigate how evolution constructs this crucial building block of intelligent behavior.

Digital evolution (Adami, Ofria, & Collier, 2000) provides the tools for exploring issues, such as evolving memory, from the bottom up and offers certain advantages over working with living organisms. Studying evolution *in silico* is much faster than working with living organisms, environmental conditions can be tightly controlled, and processes of interest can be readily isolated for study. Digital evolution is also transparent, so that we can trace everything that is happening during evolution, without influencing the system. Such an evolutionary approach can provide insights that are valuable in both biological and computational research contexts, by illuminating fundamental evolutionary issues and by finding surprising solutions to computational problems. Digital evolution, like natural evolution, may also be able to discover solutions when our intuitions as designers fail. In fact, computational evolution approaches have a track record of improving on existing computational approaches to complex problems, such as automated state diagram construction (Goldsby, Knoester, Cheng, McKinley, & Ofria, 2007), used in software requirements engineering, self-healing (Knoester, McKinley, & Ofria, 2007), and adaptive resource-aware behavior (Beckmann, McKinley, & Ofria, 2007).

1.1.2 Central Issues and Questions

Organisms must be able to respond appropriately to the environment in order to maximize their chances of survival. Aspects of their environment will often vary based on time, space, or circumstance, and organisms must react to these variations. Evolution can incorporate environmental change that is slow-paced, encompassing several generations (Nolfi & Floreano, 2000). Evolution discovered a variety of mechanisms that allow all organisms, from the simplest to the most complex, to respond to variable factors in their environments. Some of these mechanisms involve reflexive behavioral routines, such as the response of bacteria like *Escherichia coli* (*E. coli*) to move toward food, or innate behavioral preferences and patterns, as observed in many insects (Dukas & Bernays, 2000). These repertoires of fixed, innate behaviors allow organisms to successfully manage some well-defined sets of potential circumstances. However, when the variation becomes less predictable due to more possible outcomes, because of dependences on time, place, or individual organisms' experiences, or simply due to more rapid environmental changes, mechanisms with additional flexibility are needed. In the face of quicker or more unique variation, memory and learning may provide the advantage, allowing individuals to adjust behavior according to the local world state (Dukas, 2008; Shettleworth, 1998).

How do environment, memory, and learning interact in an evolutionary context? This question is of great interest on the biological side of evolving intelligence. It also has a direct connection to computer science in many ways, for example in the context of robot navigation: historically, algorithms for robot navigation have been particularly brittle when confronted with unpredictability related to locally unique features or events. Fundamental investigation focusing on the interplay between environment, memory, and learning can provide key insights for both the biological and computational perspectives.

There are many dimensions to the evolution of memory and learning, and the related questions are challenging. How memory and learning evolve is a central issue, comprising a number of interwoven questions, relating to the evolutionary pathways and building blocks of memory and learning.

In this dissertation, I investigate how evolution builds complex memory structures that can be used for learning behaviors. My approach uses evolution *in silico*, in virtual environments that present different navigation problems. To successfully solve these problems, evolution must discover mechanisms for memory and simple learning, and for differentially responding to environmental cues. With the experiments described here, I addressed four central questions:

- *What environmental conditions promote the evolution of a reflexive response to a particular cue?* This can be viewed as evolving innate "reflex" responses, *i.e.*, responses that require no learning, and do not change with experience. Although this is not a memory or learning behavior, it is a key step in two regards. First, evolving a fixed innate behavior still allows organisms to alter their responses according to features of the current environmental circumstance. Second, as seen in classical conditioning, sets of innate responses may form the basis for conditioned responses, *i.e.*, simple learning. From a high-level perspective, reflexes can be thought of as a sort of ancestral or genetic memory: evolution "hard-coded" the reflexive responses for frequently experienced, potentially harmful or beneficial environmental conditions.

- *What conditions lead to more plastic responses? Phenotypic plasticity* is the ability of a given genotype to express different phenotypes in response to differing environmental conditions. Some plastic changes are reversible, while others are not. Reversible plasticity is often considered the simplest form of learning, since it provides a capacity for altering behavior based on prior experience (Dukas, 1998).

- *What conditions lead to the transition from a set of plastic responses to the use of memory?* This question addresses the transition from a suite of innate responses, tailored to handle environmental variation, to the storage and use of individual experience to guide behavior.

- *Is it harder to evolve a life-long memory than a volatile one?* We can view this question as comparing phenotypic plasticity with more active memory

processes. Phenotypic plasticity involves evolving genetically specified traits or behavioral options, whereas more active memory processes require the ability to continually update the information and associations. From a simpler perspective, we can relate this question to mechanisms for storing information for differing lengths of time. Some information is useful for a long time, while other information might be used and discarded. What mechanisms evolve to support these different types of memory?

I use navigation as the behavior domain for the work presented here. Navigation behavior fits well with the idea of evolving behavioral intelligence, since navigation is an example of "knowing how" to manage particular problems of survival. The ability to interact efficiently and effectively with its environment is a prerequisite for an organism's survival (Shettleworth, 1998). Many tasks that are pivotal to survival, such as foraging, finding shelter, and avoiding danger, require reliable strategies for moving through the environment. Natural evolution equipped organisms with a variety of navigation strategies, reflecting the varying challenges involved in different navigational tasks (Roche, Mangaong, Commins, & O'Mara, 2005). Even fundamental navigation strategies build on other capabilities that are widely considered components of intelligence, including sensing and responding to cues in the environment, and memory. In the context of the current study, navigation is the platform for investigating how evolution builds memory structures in response to environments that challenge survival.

The current work is organized as follows. The remainder of this chapter explores background and related work in memory and learning across several different fields of study. Chapter 2 presents work related to evolving gradient following behavior and elementary memory use capabilities. Chapter 3 discusses evolving reflexive behaviors as precursors to memory and learning. Chapter 4 explores evolving one-bit memory for both short- and long-term uses. Chapter 5 presents conclusions and plans for future work.

1.2 Background

1.2.1 Memory and Learning

Definitions

The notions of memory and learning traditionally relate to separate bodies of research. Depending on the research context, memory and learning may be defined strictly, loosely, or hardly at all. The term *memory* most often refers to the process that allows animals to guide their behavior based on information from their own past experience, or to the stored record of the experience itself; memory research focuses largely on how information is stored and retrieved.

It is remarkably difficult to synthesize a representative definition of *learning.* All of these are definitions of learning:

- "Learning . . . is a change in state resulting from experience" (Shettleworth, 1998, p. 100).
- "Learning is the acquisition of neuronal representations of new information" (Dukas, 2008, p. 146).
- [Learning is] "more or less lasting changes in the innate behavioral mechanisms under the influence of the outer world" (Tinbergen, 1951, p. 143).

These definitions run the gamut from broadly inclusive to highly restrictive. Perhaps the crux of the difficulty was identified by Todd and Miller (1990, p. 307), that learning is a multi-faceted process, and "must be viewed not as a single monolithic process, but as the diverse set of distinct mechanisms, abilities, and dynamics that ethologists and psychologists have shown it to be." One common thread through these definitions is that learning refers to changes in behavior as a result of experience. As such, discussions of learning imply the existence of memory, whether over brief periods of time (short-term memory) or for more lasting durations (long-term memory) (Dukas, 2008).

Questions persist regarding the distinction between memory and learning. Shettleworth (1998) suggests that the traditional memory/learning dichotomy should be abandoned in light of the more recent emphasis on multiple memory systems. Evidence from many areas of research supports the multiple memory system hypothesis, especially in humans. The multiple memory system hypothesis is that memory is organized in different systems, according to the type of information that they mediate, and that these different types of information are processed and stored in different brain areas. Forms of memory may be characterized by how long they last, whether they involve accumulated knowledge or unique experiences, and whether memory is expressed explicitly, through conscious remembering, or implicitly, by changes in performance speed or bias (Eichenbaum, 2008). Certainly, memory and learning are intimately connected, and it is difficult to imagine the evolution of one without the other. On one hand, memory is a prerequisite for learning: an organism cannot learn from its experiences if it has no means to retain and reuse information about those experiences. On the other hand, memory cannot evolve completely free of cost, and a costly system will not evolve unless its use provides some adaptive advantage (Dukas, 1999).

Phenotypic plasticity refers to the ability of an individual organism to change from one genetically specified phenotype, which includes its behavior, to another, based on changes in that organism's environment. Dukas (1998) discusses the importance of phenotypic plasticity and its relationship to learning. Some phenotypic plasticity is irreversible, and is seen in a phenotype that is determined at one time during development, and then remains unchanged for the rest of the organism's lifetime, such as alternate wing patterns on butterflies. Other plastic traits are reversible: an individual can change those traits repeatedly in response to an environmental change. Examples of this sort of reversible plasticity are gain or loss of muscle strength according to the level of physical activity, and changes in shell thickness in the gastropod, *Littorina obtusata*, relative to exposure to predators (Trussell & Smth, 2000). Reversible plasticity implies that the organism maintains a sensory capacity to recognize a particular cue from the environment, and a mechanism to

alter something in its phenotype in response to that cue. This reversible plasticity allows adaptation when the environment changes frequently within the individual's lifetime, and permits a genetically determined association between stimuli from the environment and responses to those stimuli. Basic types of reversible plasticity may be considered as early forms of learning, since reversible plasticity provides a capacity for altering behavior based on prior experience. Learning is, itself, a complex and advanced form of plasticity. Simpler plasticity differs from more complex forms of learning in that plastic responses are still pre-determined. That is, the available responses are genetically pre-determined; which response occurs is determined by the environmental conditions. Learning, by contrast, is more open-ended: responses are not already stored, ready for use, but need to be discovered (Dukas, 1998).

Through the remainder of this work, I will use terms as follows.

- **Memory:** the mechanisms and processes of storing information about prior experience.
- **Phenotypic plasticity:** the ability of an individual organism to change from one genetically specified phenotype, which includes its behavior, to another, in response to environmental changes.
- **Learning:** "the acquisition of or change in memory that allows a subject to alter its subsequent responses to certain stimuli" (Dukas, 1998, p. 133).

Memory, Learning, and Evolution

Memory and learning play crucial roles in intelligent behavior. Even simple organisms exhibit some capacity for storing and reusing information about their environments; these capacities can be thought of as rudimentary memory capabilities. The ability of *E. coli* to compare the present and past gradient concentrations and behave accordingly is a simple form of memory (Koshland, 1979), and the slime mold *Physarum polycephalum* has the ability to find the minimum length path between two points in a maze (Nakagaki, Yamanda, & Toth, 2000). Our intuition tells us that memory and learning must often be beneficial, since they are so widespread in

nature. Nolfi and Floreano suggest that learning serves several different adaptive functions:

- Learning supplements evolution by enabling an organism to adapt to environmental change that happens too quickly to be tracked by evolution.
- The combination of ontogenetic adaptation (learning) and phylogenetic adaptation (evolution) might be able to exploit more environmental information than evolution alone.
- Learning may guide evolution, as suggested by Baldwin (Baldwin, 1896) and Waddington (Waddington, 1942). Baldwin proposed what he called *organic selection*, a process through which acquired characteristics could be indirectly inherited. Through organic selection, individuals that have the ability to acquire a beneficial trait during their lifetime will be selected for; therefore, the *capability* for acquiring that trait will be passed on to offspring. Since learning takes time, Baldwin suggested that evolution might tend to push trait acquisition to earlier points in life and thus select individuals who already have those beneficial traits at birth, eliminating the need to learn them. This indirect genetic assimilation of learned traits was later defined by Waddington as canalization, which describes developmental robustness to changes in environment and genotype.
- Learning might allow complex phenotypes to be produced from much shorter genotypes by extracting some information necessary for building the phenotype from the environment. (Nolfi & Floreano, 2000).

There is some debate regarding the selective forces that promote the evolution of learning. Stephens (1991) identified two opposing forces credited with this role—environmental change and environmental regularity—and presented a simple model that supported his argument that "the pattern of predictability in relation to an animal's life history determines the evolutionary value of learning" (Stephens, 1991, p. 77). The model takes into account the influence of three factors—change, regularity, and value—on the evolution of animal learning. Change and regularity are

measured by two terms: "between-generation persistence" describes to what extent environmental states in the parental generation predict states in the offspring's generation; "within-generation persistence" specifies how well today predicts tomorrow within the lifetime of the individual. This model emphasizes how the pattern of environmental change relates to animal's life history. Stephens stated that this model

> suggests that the most correct statement one can make is that learning is an adaptation to within-lifetime regularity and some environmental unpredictability; this unpredictability may occur either within or between generations (Stephens, 1991, p.85).

Memory and learning may also incur costs. Some of these costs may be associated with the development and maintenance of mechanisms that are precursors to memory and learning (Dukas, 1999). Other costs may include a delay in the acquisition of fitness due to suboptimal behavior during learning, increased unreliability related to the possibility of learning the wrong things, delayed reproduction time, and energy costs of the learning process itself (Mayley, 1996). Many mathematical models of memory (for example Kacelnik and Krebs (1985)) that are based on optimality modeling make the assumption that memory is cost-free and focus on how to weigh experience and what to forget. In such models, the implicit cost of memory relates to the use of irrelevant or old information, which might result in suboptimal performance (Dukas, 1999).

Even though the fitness benefits of memory and learning are relatively well understood (Dukas, 1998; Dukas & Bernays, 2000; Dukas & Duan, 2000), less is known about their costs, and evidence for the costs is difficult to come by. At present, the primary empirical evidence of the evolutionary costs of learning stems from a series of experiments by Mery and Kawecki, performed with fruit flies, *Drosophila melanogaster*. The results of one study (Mery & Kawecki, 2003) suggested an evolutionary trade-off between learning ability and competitive ability: improved learning ability in experimental fly populations was consistently associated with a decline of larval competitive ability, compared to control populations. In another study, Mery

and Kawecki (2004) forced flies to repeatedly use their learning ability by exposing the fly populations to two-day cycles of alternating substrate conditions. Flies from lines that had been selected for higher learning ability had lower egg-laying rates than flies from lines that had not been exposed to selection for increased learning ability. These results suggest that egg-laying was impaired by either the accumulation of memory interference, or the energy costs of information collection, processing, and storage. A third study (Mery & Kawecki, 2005) showed that flies subjected to training that produced long-term memory died sooner under extreme stress (absence of food and water) than flies subjected to control treatments, suggesting that the formation and maintenance of long-term memory produced added strain.

1.2.2 Memory and Learning in Insect Navigation

At the beginning of the 20th century, there was a generally held belief that insect behavior was guided nearly exclusively by instinct. The supposition was that navigation mechanisms worked because evolution had tuned the insects' sensorimotor responses and behaviors to function in the animals' typical environments, leaving insects incapable of flexibly accommodating unexpected change or of learning characteristics of particular environments (Collett, 1993). Over time, however, researchers accumulated mounting evidence of learning in various insect species. By the close of the 20th century, the weight of evidence led to the widespread acknowledgment that learning plays an integral part in the decision-making of many insects (Dukas, 2008).

Navigation behavior has provided many insights into insect memory and learning. Insects employ sophisticated navigation strategies that use a variety of environmental information. Insects, particularly bees and ants, are excellent model systems for the study of navigation: they are fascinating in their own right, and the strategies that they employ are similar to those used by birds and mammals. The more modest neural resources of insects may be analyzed more effectively to reveal essential components of navigation (Collett, 1993; Collett & Collett, 2002).

Ants, bees, and other insects use an array of innate strategies for navigation. These strategies include path integration (Müller & Wehner, 1988), which is the continual updating of distance and direction to a reference location (*e.g.*, the nest), and responses to landmarks and beacons in the environment (Graham, Fauria, & Collett, 2003; Collett, Graham, & Durier, 2003). The interaction of these strategies may reduce navigational errors and provide more robust navigational performance (Collett & Graham, 2004), and may provide a scaffold for acquiring routes between the nest and other important sites in the environment (Collett et al., 2003).

Research has provided evidence of memory and learning in ants and bees in numerous contexts related to navigation. Honeybees demonstrate mechanisms for storing motor sequences, linking motor sequences to visual stimuli, and expectations of particular visual stimuli at specific points along the route (Collett & Collett, 2002; Collett, Collett, & Wehner, 2001; Collett, Fry, & Wehner, 1993; Wehner, Michel, & Antonsen, 1996). Bees in general learn landmarks relative to the celestial compass (von Frisch, 1967), and this association can be used on overcast days to determine the compass direction (Dyer & Gould, 1981). Bees are born with a template of the solar ephemeris function (the daily pattern of solar movement through the sky), but learn the details of that function for their particular location and time of year through experience, and can estimate the location of the sun for times they have not experienced (Dickinson, 1994; Dyer & Dickinson, 1994, 1996). Honeybees associate both distant and local landmarks with previously traveled routes between the hive and food (Dyer, 1991), and ants and bees may use interconnected memories of associations between contextual cues and landmark memories to reliably recognize visual landmarks and faithfully execute learned travel routes (Collett & Collett, 2002). For long-distance navigation, honeybees depend on large features in the terrain and celestial cues but use local features to guide them to a goal (Dyer, 1998). Honeybees and other insects perform learning flights to memorize visual landmarks near a newly-discovered food source, a behavior called "turn back and look" (Lehrer, 1991). The length of these flights appears to be adjusted by the bees in response to changing visual information needs (Wei, Rafalko, & Dyer, 2002).

Studies of maze learning in animals provide insight into the mechanisms involved in navigation (*e.g.* Gallistel, 1990). Maze learning studies with insects may be of particular interest, since many bees and ants often follow fixed routes from the nest to a foraging site (Collett et al., 2003). In learning a maze, an insect is learning to follow a well-defined path.

> To follow a path means to obey a sequence of instructions. The instructions could be inherent in the external world, as when water flows down a river valley along a route imposed by the contours of the land. But the fact that bees can be trained along arbitrary routes ... and through mazes ... suggests that to some extent bees do have an internal representation of sequences of instructions (Collett et al., 1993, p. 693).

Bees have been trained to fly through mazes of varying complexity. Studies by Collett and colleagues (Collett et al., 1993; Collett & Baron, 1995) used small mazes to investigate bees' ability to learn motor or sensorimotor sequences. One study (1993) presented three different series of experiments that forced bees to fly along prescribed routes and through obstacles in a large box in a laboratory. The primary conclusion of this study is that bees remember sensory and motor information that allows them to reproduce a complex route. The bees to some degree learn a sequence of detailed instructions and do not rely solely on the external world to dictate their path. These results suggest that what is stored is a linked series of vectors, each vector a "command" for flying a certain distance in a certain direction (Collett et al., 1993).

A study by Zhang and colleagues (Zhang, Bartsch, & Srinivasan, 1996) examined whether honey bees use specific visual cues to learn to fly through structurally complex mazes. Bees were trained to fly through mazes in the presence or absence of visual cues (*i.e.*, color marks). The tested bees showed the ability to learn routes marked by a visual cue (color mark), and routes with no cues, with varying degrees of success in the different treatments. The authors report several conclusions from their results. Bees are capable of learning a complex task, such as this maze solving task, if they learn through step-by-step training, by acquiring a succession of "rules"

for negotiating the maze (*e.g.*, food is present at a location tagged with a certain color, and a path labeled with that same color leads to food). Bees can apply learned rules in new contexts, varying in spatial configuration, spectral domain (color), and both spatial and spectral domains simultaneously. While learning to follow the color marks, the bees also learn a set of motor commands for a correct sequence of actions for negotiating the maze. Bees are able to learn to navigate a maze even in the absence of internal marks (*i.e.*, visual or scent) and external marks.

A later study by Zhang and colleagues investigated bees' ability to learn and remember two different sets of visual stimuli (Zhang, Lehrer, & Srinivasan, 1999). Bees were trained to fly through a compound Y-maze. While on route to a food reward, the bees were presented with two different visual stimuli sequences. The results suggest that the bees were able to successfully store the two different sequences at the same time. Another study (Zhang, Mizutani, & Srinivasan, 2000) probed whether bees learn and recognize structural regularity in the mazes. For these experiments, bees were trained and tested in four different types of mazes: constant-turn, where turns are always in the same direction; zig-zag, where each turn alternates direction; irregular, which has no apparent pattern of turns; variable irregular, where bees had to learn several irregular mazes at the same time. The bees performed best in the constant-turn mazes, somewhat poorer in the zig-zag mazes, still worse in the irregular mazes, and poorest of all in the variable irregular mazes. The authors state that these results demonstrate that the bees' performance in the various configurations depends on the structural regularity of the mazes, and the ease with which the bees can recognize and learn that regularity.

Bees provide an example of behavioral intelligence that can inform my experimental design, and so maintain a strong connection between my experiments and their biological motivations. I based my experimental environments on selected maze configurations from the maze-learning experiments with bees, discussed above. As highlighted in the preceding discussion, bees use a variety of navigation strategies that demonstrate use of different memory capabilities. Maze environments allow for systematic manipulation of environmental features to probe aspects of memory

use and mechanisms, and are easily constructed in virtual settings. By using these types of environments in the work presented here, I am able to probe specific issues relating to the evolution of memory.

1.2.3 Computational Approaches to Memory, Learning, and Evolution

In the context of computer science, the concepts of memory and learning take on particular connotations. The term "memory" is linked primarily to hardware-related issues, and the more specific "associative memory" points to specific neural-network based implementations, often combined with self-organization (Kohonen, 1977, 2001). Machine learning is an active and rich research area, encompassing a variety of methods, including supervised learning (Kotsiantis, 2007, review), unsupervised learning (Kotsiantis & Pintelas, 2004, review), and reinforcement learning (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998, review). Machine learning focuses on how to engineer machines that learn, or behave in a manner that resembles learning, as opposed to evolving learning machines.

Generally speaking, evolutionary computation employs algorithmic methods inspired by Darwinian natural selection to find solutions to computational problems. Traditional evolutionary computation, comprised of such areas as evolutionary algorithms, evolutionary strategies, genetic algorithms and evolutionary programming, is largely concerned with optimization problems. These approaches also tend to be highly applied, with a focus on using evolutionary processes in solving particular instances of problems, in contrast to investigating how general solutions arise through evolution (De Jong, 2006).

Other approaches are more concerned with evolving machines that exhibit some life-like behavior. Evolutionary robotics is one such methodology. Evolutionary robotics refers to efforts to develop robots that autonomously evolve and adapt to their environments (Floreano, Mondada, Perez-Uribe, & Roggen, 2004). This strategy emphasizes the importance of embodiment, autonomy, and adaptation. In this

technique, adaptation most often occurs as evolution in neural network controllers through the use of genetic algorithms (Floreano, 1997). Although not truly within the realm of evolutionary robotics, Valentino Braitenberg's (1986) work could be considered a precursor of that field. In these thought experiments, Braitenberg envisioned a series of simple wheeled robots, with different kinds of sensors that are connected in various ways to the wheel motors. When the robots are placed on a table, they exhibit different behaviors, such as approaching lights, running away from lights, and remaining close to the light. In the course of traveling around the table top, some robots will fall off the table, making it necessary to build copies of the robots in order to maintain a given number of robots on the table. In copying the robots, the builder is bound to make mistakes, resulting in new designs and potential improvements over time. This process of selective copying and random errors is clearly Darwinian evolution, and Braitenberg's delightful thought experiments provide an excellent jumping-off point for later work in evolving actual robot behavior.

The field of evolutionary robotics has dealt extensively with several facets of evolving memory and learning. One aspect is phenotypic plasticity, the ability of a genotype to express differently in different environments. Nolfi, Miglino, and Parisi (1994) studied this topic by evolving neural network "brains" for virtual robots in environments that alternated between light and dark; the dynamics of the environments were designed such that behaviors that were successful in one environment would be unsuccessful in the other environment. Individuals that evolved under these conditions were able to tune their behavior appropriately for both kinds of environments, adapting within an individual "lifetime" to environmental changes.

Although not within the scope of robotics, a closely related study by Stanley, Bryant, and Miikkulainen (2003) used NeuroEvolution of Augmenting Topologies (NEAT) to compare neural networks that never changed connection weights to those that did. In this experimental environment, food switched randomly between nutritious and poisonous. The fixed-weight networks found solutions that worked for both environmental conditions: different inputs to the networks produced different

behaviors.

There is increasing interest in the evolution of learning and the interaction between learning and evolution within the evolutionary robotics community. This interest is spurred by several purposes that are shared by other approaches and methodologies. One purpose is to examine the performance advantages of combining evolutionary adaptation and learning; another purpose is to understand the role played by the interaction of these two adaptive processes, which employ different mechanisms and occur at differing time scales (Nolfi & Floreano, 2002). A study by Floreano and Urzelai (2000) is a strong example of the latter. They evolved neural networks with local synaptic plasticity and compared them to fixed-weight networks in a two-step task. The networks evolved to turn on a light and then move to a grey square. The results showed that local learning rules helped networks alter functionality quickly, facilitating moving from one task to the other. Blynel and Floreano (2003) explored the ability of continuous time recurrent neural networks (CTRNNs) to show capabilities that resemble reinforcement learning, in the context of T-Maze and double T-Maze navigation tasks. The robot had to find and "remember" the location of a reward zone. The learning in this case occurred without modification of synapse strengths, coming about instead from internal network dynamics. This work was directly related to an earlier study by Yamauchi and Beer (1994), that investigated the evolution of agents capable of combining reactive, sequential, and learning behavior, using CTRNNs to control their agents. A study by Todd and Miller (1990) explored the conditions under which simple artificial creatures are more likely to evolve learning mechanisms for differentiating edible from poisonous food. In this model, learning was a particular connection between a color sensor and the single motor neuron of a neural network, and the genetic specification that controlled the development of the learning structure was specified by a single gene. Tuci *et al.* (2002) applied evolutionary robotics methodology to studying the evolution of learning behavior from an ecological perspective, which treats each instance of learning as a specialized capability that is shaped by selective pressures; from this perspective, learning can be understood only by reference to the organism or

its ancestor. Their model required a robot to learn the relationship between a light and the location of its target. The robot had to interact with its environment to learn the relationship between the light and the target; in one environment, motion toward the light took the robot toward its target, but in another environment, the relationship between light and target was inverted. Results of their experiments show that artificial evolution can be used to combine low-level building blocks to produce controllers that are capable of associative learning.

1.2.4 Avida: Overview

The Avida Digital Evolution platform (Lenski, Ofria, Pennock, & Adami, 2003; Ofria & Wilke, 2004) is a widely used digital evolution software system. Digital evolution (Adami et al., 2000) is a form of evolutionary computation that places a population of self-replicating computer programs—"digital organisms," or "Avidians"—in a user-defined computational environment without an explicit fitness function. Avida provides a virtual environment, but real evolution occurs: the digital organisms self-replicate, mutate, and compete, satisfying Dennett's definition of an evolutionary process (Dennett, 2002). Digital evolution can be used as a tool to provide a better understanding of biological processes, and as a method for applying lessons learned from biology to computational and engineering problems. Avida affords the opportunity to study evolution in detail, and look "inside" the process as it happens. These are both difficult things to achieve with a natural organism, even one as simple as a bacterium. Since Avida is another instance of evolution, we can use Avida for generalizations about evolution. According to John Maynard Smith (1992),

> So far, we have been able to study only one evolving system, and we cannot wait for interstellar flight to provide us with a second. If we want to discover generalizations about evolving systems, we will have to look at artificial ones (Maynard Smith, 1992, p. 772).

Pennock (2007) discusses these issues in detail.

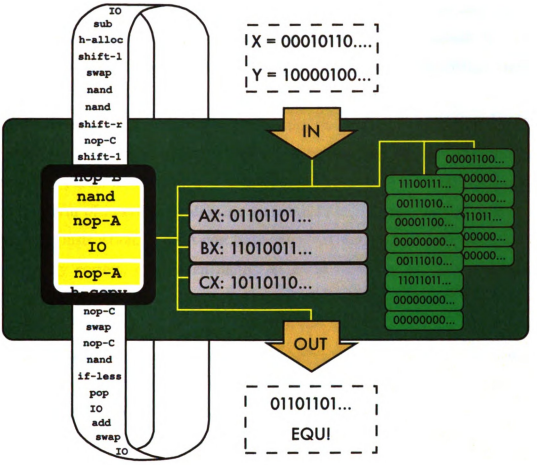


Figure 1.1: An Avida digital organism. The organism comprises a circular genome, a virtual CPU with three 32-bit registers, two stacks, and four heads (FLOW, IP, READ, WRITE). Numerical values can be input from the environment, and results are output to the environment. (Lenski et al., 2003, p. 139)

The Avida world is a discrete two-dimensional grid of cells, containing a population of individual digital organisms, with at most one organism in each grid cell. Each individual organism (Figure 1.1) comprises a circular list of assembly-like instructions, its "genome," and a virtual central processing unit (CPU). The virtual CPU consists of three general purpose registers, two stacks, and four heads (FLOW, used as a jump target; IP, an instruction pointer that denotes the next command to be executed; READ; and WRITE). Executing the instructions in the organism's genome acts on the elements of the virtual CPU, incurring a cost measured in virtual CPU cycles. Executing Avida instructions accomplishes all functions in the Avida

world, such as gathering information from the environment or performing logic operations. The Avida instruction set is Turing-complete (Ofria, Adami, & Collier, 2002) and extensible, affording ease in expanding the system's capabilities through adding new instructions.

An Avidian replicates by copying its genome into a block of memory that will be its offspring's genome. Errors in the copying process produce differences between the genomes of parent and offspring. These differences are *mutations*; these mutations take the form of inserting or deleting an instruction, or changing one instruction to another instruction. The Avida instruction set has the property of remaining syntactically correct, even when mutations occur (Ofria et al., 2002). A newly produced offspring is placed into a randomly selected grid cell, overwriting any organism occupying that grid cell. This gives an adaptive advantage to organisms that can replicate more quickly: the organism must compete for the limited space in the grid. Organisms that replicate sooner than others will have a higher proportion of descendants in future populations. Avidians can speed up their execution, and so replicate sooner, by accumulating "metabolic rate" bonuses by performing user-specified tasks. Metabolic rate is used to allocate virtual CPU cycles. Organisms with higher metabolic rates are given more virtual CPU cycles in a unit of time than organisms with lower metabolic rates, and so are able to execute more quickly and produce offspring sooner.

Each Avida organism has a facing, *i.e.*, the direction in which it is oriented. An organism must always have a valid facing, meaning that it must face a cell that is connected to the organism's cell. Because of differences between the geometries available in Avida, there are varying numbers of valid facings in certain geometries. The bounded grid geometry creates a world with defined "edges." Grid cells at the edges have no connection to the cells outside the grid boundary. In a bounded grid geometry, most grid cells have eight valid facing directions (North, Northeast, East, *etc.*, Figure 1.2); grid cells that are along the edge or in the corners of the grid have fewer valid facing directions (three in corners, Figure 1.3, and five at edges, Figure 1.4). A torus geometry creates an infinite plane, with no edges. In a torus

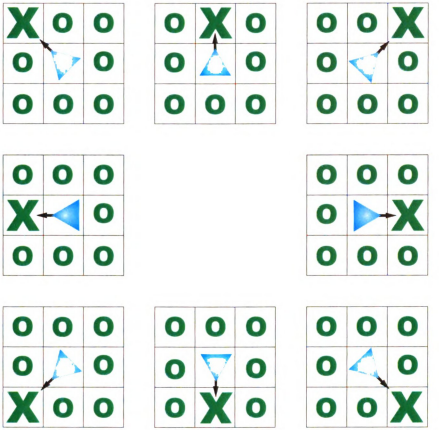


Figure 1.2: Avida facing, torus or bounded grid interior. Valid facings for an Avida organism in a torus or in the interior (*i.e.*, non-edge) cells of a bounded grid. The organism may be oriented toward any of its eight neighbor cells.

geometry, all grid cells have eight valid facing directions (Figure 1.2).

An Avida organism may use the *move* instruction to move from its current grid cell into the cell the organism is currently facing. If another organism is occupying the destination cell, the two organisms exchange places, and their previous facings are preserved. If an organism moves in such a way that it faces across a grid edge (on a bounded grid), the organism's facing is automatically changed, by rotating the organism to face the next valid clockwise facing.

Figure 1.3: Avida facing, bounded grid corner. Valid facings for an Avida organism at the corner of a bounded grid. Only three valid facing directions exist in bounded grid corners.

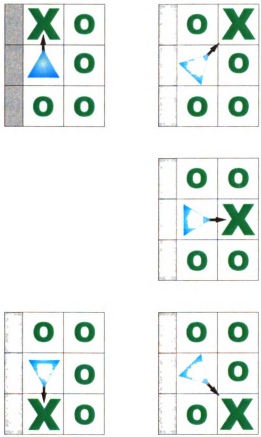


Figure 1.4: Avida facing, bounded grid edge. Valid facings for an Avida organism at the edge of a bounded grid. Facings toward the grid edge are invalid; the organism's facing is automatically adjusted so that it never faces off the edge of the grid.

Chapter 2

Evolving Rudimentary Memory

Studies of capabilities such as memory and learning often presuppose the existence of some neural structure, however modest. But the evolutionary roots of these capacities may predate even simple brains. To address the evolutionary origins of memory and learning, we need to step farther back in evolutionary history, and farther down in complexity. Briggman *et al.* (Briggman, Arbanel, & Kristan, Jr., 2005) propose that the idea of decision-making applies to a "spectrum of goal-driven behaviors," from simple, predictable reflexes to conscious choices made with expectation of specific outcomes. We can apply this same thinking to our investigation of the evolution of memory and learning, by looking to the simplest forms of those abilities that we can identify. The decision-making behavior of bacteria, specifically *E. coli*, provides a point of departure. There are a number of reasons for using *E. coli* as the jumping-off point for studying decision-making and the evolution of memory and learning: *E. coli* is simple, studied extensively, and known to obtain and use environmental information in its survival strategies.

One particular type of simple decision-making is the ability of cells and organisms to orient relative to sensory stimuli. Simple cells and microorganisms exhibit orientation responses to a variety of environmental stimuli. A distinction is often made between *taxes* (singular *taxis*, meaning tactic response), responses of cells that are capable of locomotion, and *tropisms*, in-place responses of cells that are attached to a substrate. Taxes are positive or negative, involving movement toward

a stimulus or away from a stimulus, respectively. The type of stimulus provides a basis for classifying taxes, for example chemical agents (*chemotaxis*) or light (*phototaxis*). Orientation mechanisms serve to move organisms toward conditions that are somehow beneficial, or away from conditions that may be harmful (Carlile, 1975).

Chemotaxis in bacteria may be an ancient process. The process may predate the divergence of the eubacteria (gram-negatives, gram-positives, and blue-green algae) from the archebacteria (methane bacteria), since chemotactic behavior is evident in both kingdoms (Ordal, 1980). T. W. Engelmann and W. F. Pfeffer first described bacterial chemotaxis in the late 19th century (Adler, 1975; Berg, 2004), and pioneering researchers such as Julius Adler revealed its molecular mechanisms beginning in the 1960's (Adler, 1966; Berg & Brown, 1972). Chemotaxis involves bacteria's differential reaction to gradients of substances in the environment: the bacteria tend to move toward certain substances (attractants), and away from other substances (repellants). *E. coli* use a simple strategy for motility, moving in response to attractants or repellants in the environment. The bacteria's movements consist of more or less straight-line swimming, usually called a "run," punctuated by random thrashing that results in a new orientation, usually termed a "tumble." In a chemically uniform environment, the movement is essentially a random walk; when a cell is moving through increasing concentrations along an attractant gradient, however, tumbling occurs less often, producing a biased random walk that allows the bacteria to move toward the attractant source (Adler, 1966; Berg, 2004). In short, *E. coli* tend to swim in the same direction for longer periods of time when the situation is improving (*i.e.*, they are moving uphill on the attractant gradient), but return to the random walk otherwise (Berg, 2004).

Since *E. coli* are capable of following a chemical gradient, they must, in some fashion, sense environmental information and make comparisons between separate sensory values. This separation could be spatial, using simultaneous readings from spatially separated sensors (*e.g.*, sensors at the front and back of the cell), or temporal, using samples from a single sensor taken at different times. *E. coli* cells are relatively small, and evidence indicates that they use temporal sensing rather

than spatial (Berg, 2004). Since the cells can compare present and past values and respond relative to that comparison, some researchers regard this capability as a simple "memory" (Koshland, 1979).

2.1 Evolving Intelligent Tactic Response and a One-timestep Memory

We have reported results for evolving a taxis-like response, and a small memory use capability (Grabowski, Elsberry, Ofria, & Pennock, 2008). In this study, we focused on the *de novo* evolution of motility and tactic response, inspired by the chemical gradient-following behavior of organisms like *E. coli*, as described above. The purpose of this study was to provide proof-of-concept for motility in the Avida system, and evolving simple navigation capabilities.

2.1.1 Orientation and Movement in Avida

As with other aspects of an Avida organism's execution, orientation in Avida is controlled through Avida instructions. A variety of rotation instructions change an organism's orientation in different ways. For example, *rotate-right-one* and *rotate-left-one* change the organism's facing to the next valid facing in the specified direction, and *rotate-label* changes the facing by a multiple of 45 degrees, determined by a NOP label that follows the instruction. In all rotation instructions, only valid facings are used for the new orientation; any invalid facings are ignored, and the next valid facing is chosen instead (see section 1.2.4 for details about valid facings).

In these experiments, the implementation of movement is based on the tumble-and-run chemotaxis behavior as observed in bacteria like *E. coli*, described earlier. We added two instructions to provide motility, *move* and *tumble*. The *move* instruction moves the organism one grid cell in the direction that the organism is currently facing. If the destination cell is occupied by another organism, the two organisms simply exchange cell locations on the grid. To allow room for organisms to move

around the grid, we added a population cap that limits the size of the total population to a user-defined number of organisms. When the cap is reached, organisms are "killed" at random when a new organism is added. We added the *tumble* instruction to emulate the random tumbling orientation behavior of *E. coli*. When an organism executes *tumble*, the organism is given a facing (orientation) at random from the available valid facings.

2.1.2 Experiments and Results

In our experiments, we demonstrate the evolution of a chemotaxis-like response. This response associates a feature of the environment (a gradient) with an advantageous outcome (a metabolic rate bonus). Two sets of experiments comprised the study. The first set of experiments focused on the evolution of tactic response under two different memory treatments, one built in and one evolved. We implemented two sensing instructions for the different memory treatments: one instruction provide a built in, or implicit, memory, by giving the organism the current and the previous sense information; the other instruction provides only the current sense information (see below for more details of the sense instructions). The second set of experiments compared two different rotation strategies, one systematic and the other random. We used the *tumble* instruction (described above) for the random rotation strategy and an existing Avida rotation instruction, *rotate-right-one* for the systematic rotation (see below for additional information about the rotation instructions).

Idealized Gradient and Sensing

Environments in the real world are inherently complex. For example, gradients are subject to multiple kinds of dispersal, such as turbulence and diffusion. Living organisms' behavior in these complex environments can be difficult to understand, so controlling for such factors is an important aspect of experimental methodology in the study of microorganisms (Adler, 1975). The virtual world of Avida provides the opportunity to abstract the essential features of the environment, to better elucidate the behavior of the digital organisms.

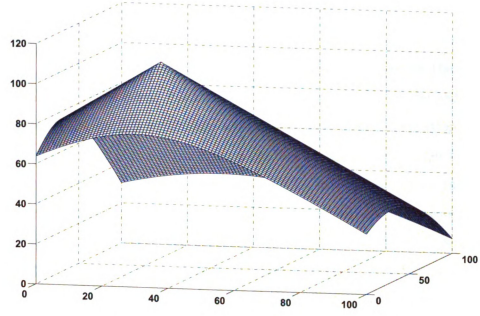


Figure 2.1: Example of distance as an idealized gradient. The "concentration" source location is at the top of the hill. Note that it is not necessary for the peak to be in the center of the environment (from Grabowski et al., 2008).

The idealized gradient is an abstraction of a physical gradient, using an easily-calculated quantity to establish the environment's gradient, removing the need for complex and computationally expensive artificial physics. An Avidian "senses" the gradient by using instructions that were added for these experiments. The instructions function as "black boxes": the digital organism has access only to the output of the instruction, not to any information that is used within the instruction's implementation. The process of producing the information is impenetrable to the Avidian, much the same way as the computations within our own brains are hidden from us. This detail makes it possible for us to implement instructions that allow a digital organism to "sense" any salient feature or property of the virtual environment without giving them access to global information.

The idealized gradient for these experiments was formed by the Euclidean distance between two grid cells, the organism's location and a "target" location. "Target" is used here in the sense of a source of a sensory stimulus, *i.e.*, a chemical

attractant (Figure 2.1). The distance between the current and target locations is analogous to the concentration of a particular attractant that the organism will move toward. The sense value provides strictly local information. The organism never has access to any global information, such as (x, y) coordinates; it senses the current gradient concentration (squared distance to target) as a simple integer value. All calculation details are hidden within the sense instruction's implementation.

At birth, each digital organism is given a randomly determined target cell, at a user-defined minimum distance from the organism's initial location. Since the target grid cell is analogous to an attractant source, the instruction that was implemented to "sense" the distance between the organism and its target grid cell is equivalent to an idealized sensory system. In biological systems, both the sensory system and the organism's interest in the particular attractant would have evolved. Just as we start experiments with a self-replicating ancestor, since we are not investigating origin of life issues, we can allow the sensory system and the effect of the attractant to be available at the beginning in these experiments, since the evolution of those mechanisms was not relevant to our central concern of evolving motility and tactic response.

We used two sensing treatments for these experiments: one treatment provided the Avidians with an implicit memory system to remember previous gradient concentrations; the second treatment removed this crutch, requiring the organisms to evolve a mechanism that would allow them to store and reuse previous values on their own. We used a different sensing instruction in each treatment. One instruction (*sense-and-remember*) placed both the current distance information and the previous distance information into two of the organism's virtual CPU registers, providing a small implicit memory. The second instruction (*sense-now*) did not provide implicit memory; this instruction placed only the current sense value into a register, requiring the organisms to discover a mechanism for storing the prior distance information. For both sensing instructions, the organism must use additional instructions to process these values. For example, comparisons must be performed in addition to the sense instruction.

Table 2.1: New Avida instructions used for experiments.

Experiment	Instructions used
Taxis, implicit memory	*sense-and-remember*, *move*, *tumble*
Taxis, evolved memory	*sense-now*, *move*, *tumble*
Rotation strategy, random	*sense-and-remember*, *move*, *tumble-and-move*
Rotation strategy, systematic	*sense-and-remember*, *move*, *rotate-right-one-and-move*

Another set of experiments compared the efficacy of two different orientation strategies, in the setting of the idealized gradient. In these experiments, we compared the effectiveness of the random *tumble* instruction to the existing *rotate-right-one* instruction, that changes the organism's facing by one 45-degree clockwise turn. We implemented specialized instructions to minimize the possibility for organisms to execute multiple rotation instructions before moving. These instructions combined the rotation instruction (*tumble* or *rotate-right-one*) with a *move* immediately following. These combination instructions compelled organisms to move immediately after turning, preserving the distinction between systematic and random rotation strategies. The rotation strategy experiments used only the implicit memory sensing treatment, as described above, with the *sense-and-remember* instruction. Table 2.1 provides a summary of the new instructions used for each set of experiments.

Experimental Setup

We seeded the population of each Avida experiment with a simple self-replicating organism (an organism with only the capability to replicate, *i.e.*, make a copy of itself). This seed organism's genome comprises 100 instructions, composed of a short copy loop and a large number of no-operation instructions. Other instructions can appear only through mutation. All experiments used the default Avida mutation rates, with a 0.0075 copy-mutation probability per copied instruction, and insertion and deletion mutation probabilities of 0.05 per divide (overall 0.085 genomic mutation rate for a length-100 organism) (Ofria & Wilke, 2004).

We placed each population in a 100×100 cell bounded grid, with a population cap of 1,000 organisms, to allow room for organisms to move around the grid. We

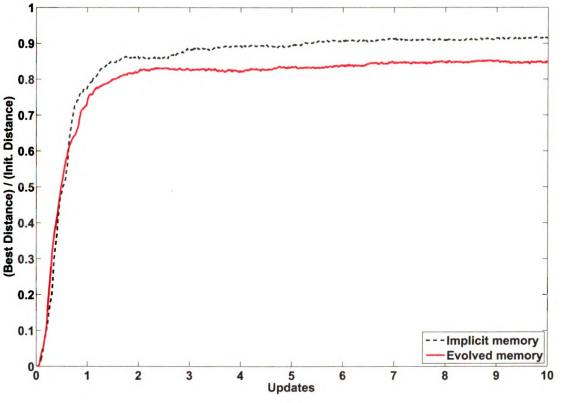


Figure 2.2: Average best distance ratio for evolution of taxis experiments using *sense-and-remember* instruction (both the current and the previous sensed value are given to the organism, but comparisons must be performed in addition to this instruction) and *sense-now* (the organism must remember previously sensed values and perform comparisons in addition to this instruction). The average represents all organisms in the population at each update (total of approximately 1,000,000-2,000,000 total organisms per run), averaged over all 50 replicates of each treatment (Grabowski et al., 2008).

ran the experiments for 100,000 updates, the natural unit of time in Avida (in these experiments, 100,000 updates are approximately 10,000-15,000 generations), with 50 replicate populations in the memory treatments experiments, and 100 replicate populations in the rotation strategy experiments.

Results

The primary performance metric in these experiments is the *best distance ratio*. The ratio is computed as $1 - (d_b/d_i)$, where d_b is the distance value of the organism's

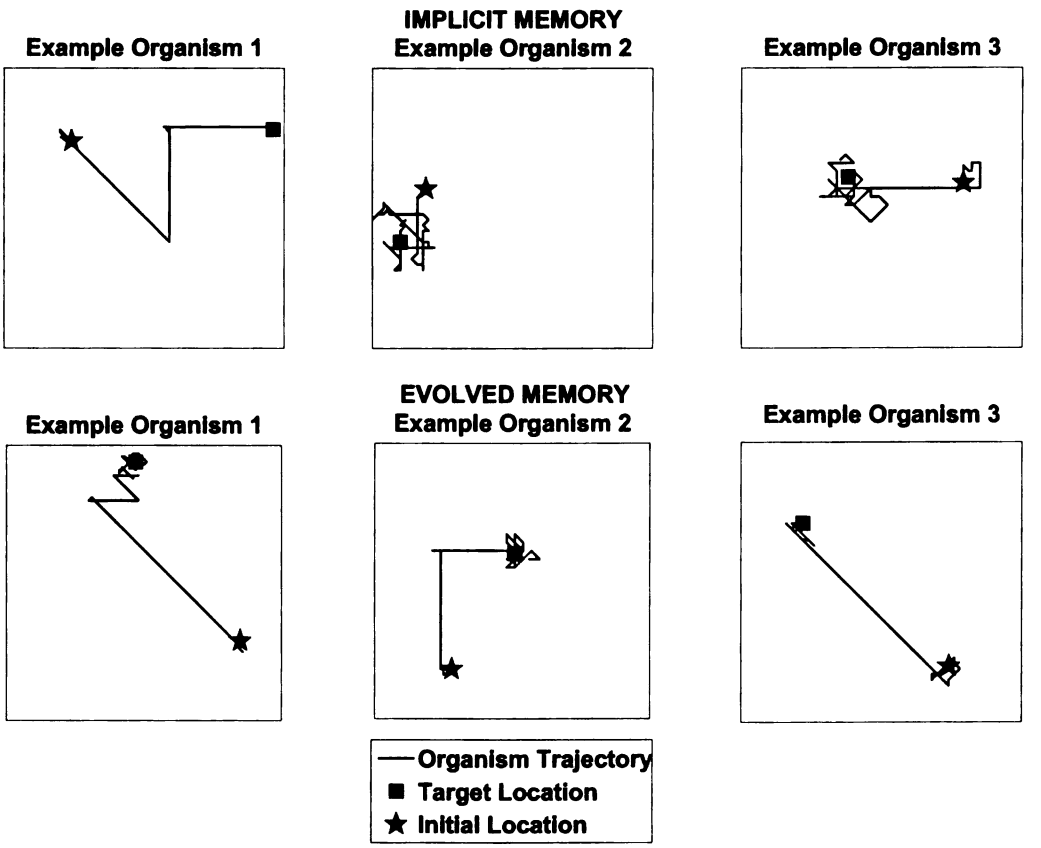


Figure 2.3: Sample trajectories of evolved organisms using implicit memory (top) with the *sense-and-remember* instruction (both the current and the previous sensed value are given to the organism), and evolved memory (bottom), using the *sense-now* instruction (the organism must remember previously sensed values). All rotation uses the *tumble* instruction, and the organisms must use additional instructions to perform comparisons. The trajectories suggest that the organisms are using the sensed information to track to their target locations.

closest approach to its target (the best distance to target), and d_i is the distance between the target and the organism at birth (the initial distance to target). Avida tracks and stores the best distance ratio for each organism, and computes the population's average best distance ratio for each update. This statistic actually reports the average best distance ratio as of the last divide (*i.e.*, the parents of the current population). This technique serves to reduce biasing caused by such factors as the age structure of the current population.

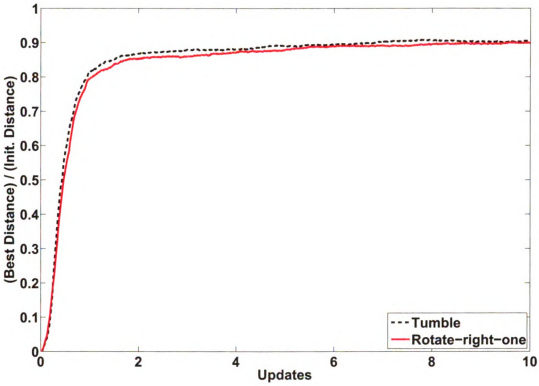


Figure 2.4: Average best distance ratio for *tumble-and-move* and *rotate-right-one-and-move* instructions. The figure shows the average best distance ratio over the entire population, for 100 replicates of each instruction set. Experiments ran for 100,000 updates (approx. 10,000-15,000 generations) (from Grabowski et al., 2008).

As illustrated in Figure 2.2, tactic behavior evolved successfully, both with and without the use of implicit memory. The behavior emerged quite quickly, within the first one to two thousand updates. The speed with which motility emerged is demonstrated by the average best distance ratios over all 100,000 updates for the two treatments, 0.8396 for taxis with implicit memory and 0.7910 for taxis without implicit (*i.e.*, with evolved) memory.

The average best distance ratio for the last 50,000 updates shows the overall success of the evolved taxis strategies, 0.9028 using implicit memory, and 0.8480 with evolved memory. These highly successful strategies emerge quickly and become relatively stable within the first 50,000 updates. Although both treatments produce successful solutions, the use of implicit memory produces significantly better performance, as shown by a Mann-Whitney U-test comparing the average best

distance ratios at the ends of each experimental run ($N = 50$, $p = 4.93 \times 10^{-7}$). Figure 2.3 shows representative trajectories of evolved organisms for each of the two treatments.

In this simple experimental environment, the difference in the performance of the random (*tumble*) rotation strategy and the systematic (*rotate-right-one*) rotation strategy was not significant (Mann-Whitney U-test, $N = 100, p = 0.286$). As shown in Figure 2.4, the average best distance ratios in the last 50,000 updates were similar, 0.900 for the *tumble-and-move* treatment, and 0.893 for the *rotate-right-one-and-move* treatment. This result shows that, at least in this discrete, idealized environment, both the random and systematic rotations strategies perform "well enough." We suggest that this result serves as a simple demonstration of "satisficing": given a particular set of constraints, the process of evolution will discover solutions that work adequately under those constraints. Satisficing is a term coined by Herbert Simon, originally in the context of economics. Satisficing refers to a decision-making strategy that meets criteria of adequacy rather than what might be considered optimality. This relates to the idea of bounded rationality, taking into account both the limitations of knowledge and the limitations of cognitive capacity in decision-making (Simon, 1957). The notion of satisficing fits well with evolutionary processes. As human observers, we may think that there is a "better," more "optimal" solution to a particular problem. Evolution will choose a "better" solution when such a solution is discovered, but may move through many "good enough" solutions on the way to the optimal solution. Movement toward such a "better" solution will occur in response to changing selective pressures, or by chance mutations (Grabowski et al., 2008).

Table 2.2: Memory treatments and simulated noise levels tested.

Memory treatment	Noise levels (±%)
Implicit memory	10, 20, 50, 75, 100 (*sense-random* instruction)
Evolved memory	10, 20, 30, 40

2.2 Further Experiments: Robustness to Sensor Noise

2.2.1 Methods

I conducted additional experiments to test the robustness of the underlying evolved gradient following algorithms. To accomplish this, I added uncertainty in the form of noise. There are three possible sources of noise in this context: the environment, meaning the environment itself is variable, the sensors, so that there is inherent error in the operation of the sensors, or the effectors, such that there is error in the way that the effectors carry out motor commands. In the natural world, all of these may happen simultaneously. For the purpose of these experiments, I chose to introduce simulated sensor noise, while maintaining a stable environment and correct execution of movement instructions.

To simulate sensor error, I implemented new sensing instructions that added varying amounts of "noise" into the gradient concentration sensor value (*i.e.*, the distance to the target cell) by altering the value of the sensed concentration (distance to target cell) by a randomly selected amount within a user-defined range. The sensed distance (SD) value was calculated as $SD = TrueDistance * (1 \pm level)$, where $TrueDistance$ is the actual current distance to the target location, and $level$ is the randomly selected amount of noise. The sensed distance could, therefore, be either less or more than the actual distance. I implemented two instructions that used this approach, one that used implicit memory (*sense-and-remember-noisy*) and the other requiring evolved memory (*sense-now-noisy*), as described previously in section 2.1.2. A different instruction, *sense-random*, returned a completely random value for the sensed distance to the target location. Instead of altering the current

distance value, this instruction generated a random number and used it as the sensed distance value. The value could be anywhere in the range from 0 to the size of the world. This method maximized the possible sensor error by simulating a completely unreliable sensor that conveyed no information. This instruction used only the implicit memory treatment. Table 2.2 summarizes the simulated noise levels tested for each memory treatment.

The size of the world, mutation rates, population size, and length of experimental runs were the same as in the previous idealized gradient experiments (section 2.1.2).

For the sake of comparison, I implemented a hand-coded organism that performed a random walk. This organism used the instruction set from the evolution experiments described above. Instead of using any sensory information, the hand-coded organism chose randomly between moving and tumbling at each time step. Tracing this organism's trajectory provided an example of random movement, to serve as a baseline for comparison with the evolved strategies.

2.2.2 Results

As in the earlier experiments (section 2.1.2), I used the average best distance ratio as the primary performance metric. In related work (Elsberry, Grabowski, & Pennock, 2009) we demonstrated that, in an Avida bounded grid environment, an information-free movement strategy will allow an organism to approach a target location (a resource concentration peak, analogous to the target cell location in the current experiments) to within 20% of the length of the longest side of the grid. Following from this information, I used a threshold of the average best distance ratio in order to classify replicate runs as having evolved tactic behavior. A best distance ratio of 0.90 or higher indicated that the population had evolved gradient-following behavior. This is a reasonable threshold value: since the average best distance ratio measures the performance of the entire population, not just an individual organism, it is likely that populations with ratios at or above the threshold value are, on average, using a gradient ascent strategy. It is unlikely that enough organisms in the population are "lucky" enough to approach their targets so closely with a random or information-

free strategy. Since the sample size was small for the simulated noise runs ($N = 20$), I tested statistical significance using Fisher's Exact Test.

Some replicate populations in both the implicit and evolved memory treatments evolved gradient following behavior under low and moderate levels of noise. At the ±10% noise level, the proportion of runs that evolved gradient climbers was not significantly different from the noise-free conditions (implicit memory treatment, $p = 0.5083$; evolved memory treatment, $p = 0.1357$). Higher levels of noise resulted in significant differences in the evolution of gradient ascent strategies in noisy and no noise conditions. With ±30% noise, the evolved memory condition showed a significant difference between noisy and no noise conditions in evolved taxis ($p = 0.0440$), but there was no significant difference at ±40% ($p = 0.2880$). It is possible that this non-intuitive discrepancy between the results at ±30% and ±40% levels may disappear if the sample size or the length of experimental runs were increased. With the implicit memory condition, noise and noise-free conditions produced significantly different distributions in the populations at levels of ±50% and above (±50%, $p = 2.338 \times 10^{-6}$; ±75%, $p = 1.805 \times 10^{-7}$; ±100%, $p = 6.868 \times 10^{-9}$).

We can see some interesting qualities in the evolved strategies by looking at trajectory plots for representative evolved organisms. I tracked trajectories for a single organism per run during 100 updates, with mutation rates set to zero; the tracking run terminated if an organism replicated before the 100 updates elapsed. Figure 2.5 shows example trajectories of the top three organisms using implicit memory (*i.e.*, the three highest Best Distance Ratio (BDR) values), with and without ±10% simulated sensor noise (Figure 2.5). The character of the organisms' movement track is similar in both noisy and no noise conditions. With implicit memory, even at much higher noise levels (*e.g.*, ±50% and ±75%, shown in Figure 2.6, implicit memory only) some organisms' algorithms seem to make attempts at using the unreliable sensor information, producing trajectories that resemble a biased random walk. The picture is different for the evolved memory treatment (Figure 2.5). Only 1 of 20 replicate populations finished the run with a BDR of 0.90 or higher (organism tra-

jectory trace shown in bottom row, left-most plot in Figure 2.5). The organisms that produced the other two plots shown in the bottom row of Figure 2.5 are not gradient followers. Instead, they use an information-free strategy that exploits the bounded grid geometry (see more detailed discussion, following).

At the highest noise level (*i.e.*, using the *sense-random* instruction, which returns a random value as the sensor value), the organisms do not use the information at all, relying instead on an information-free algorithm that provides adequate performance in the bounded grid environment, as discussed in Elsberry *et al.* (2009). It is important to note, however, that this strategy is not itself random. Figure 2.7 illustrates trajectories of a hand-coded random walker organism and an organism using the *sense-random* instruction. The spatial patterns of the two strategies are markedly different. The information-free strategy in the *sense-random* instruction organism evolved to leverage the bounded grid geometry by exploiting the built-in facing behavior at the edges of the grid. As previously noted, an Avidian must always have a valid facing. When an organism reaches the edge of the grid, its facing is automatically changed to the next valid facing such that it never faces across an edge (see section 1.2.4 for more details about facing). This characteristic of the geometry allows an organism to "run" continuously, with occasional forays across the grid, typically on one of the diagonals. In this way, an organism can run near to its target cell location and accumulate a higher metabolic rate bonus, without ever sensing the target. In contrast, a true random walk may or may not approach a target location within a finite amount of time (even though such an algorithm is guaranteed to hit any location in two dimensions, given unlimited time).

These results demonstrate remarkable robustness in the evolved gradient following algorithms, even with significant simulated sensor noise (±20%). This outcome agrees with results of other studies (*e.g.*, Nolfi and Floreano (2000)). The information-free strategies evolved in the maximum noise condition (*sense-random* instruction) also show remarkable adaptation to the environment by successfully exploiting the bounded grid geometry.

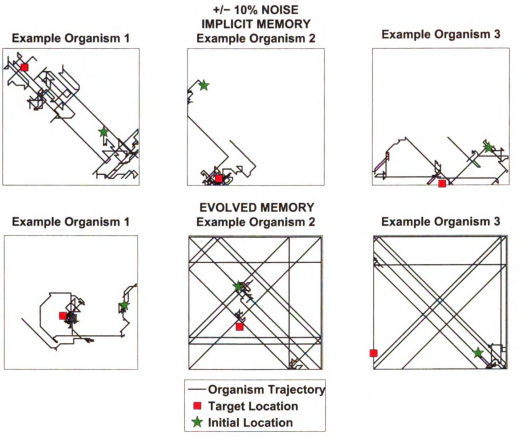


Figure 2.5: Sample trajectories of organisms from different memory treatments, $\pm 10\%$ simulated sensor noise. Trajectories shown for the most abundant genotype at the end of evolution runs for the replicate populations with the three highest Best Distance Ratios for the two memory treatments (implicit memory, top: both the current and the previous sensed value are given to the organism; and evolved memory, bottom: the organism must remember previously sensed values). All rotation uses the *tumble* instruction, and the organisms must use additional instructions to perform comparisons.

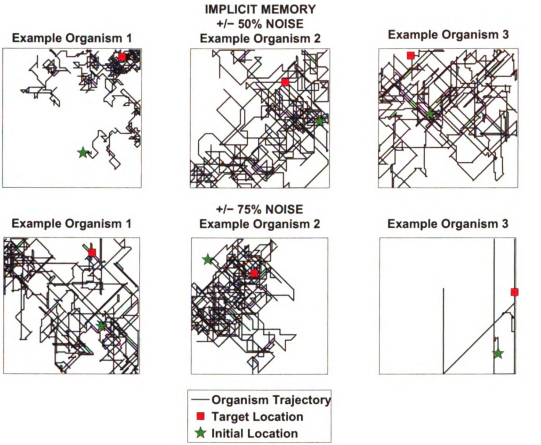


Figure 2.6: Sample trajectories of organisms from implicit memory treatment, ±50% and ±75% simulated sensor noise. Trajectories shown for the most abundant genotype at the end of evolution runs for the replicate populations with the three highest Best Distance Ratios for the two noise treatments (±50%, top, and ±75%, bottom). All rotation uses the *tumble* instruction, and the organisms must use additional instructions to perform comparisons.

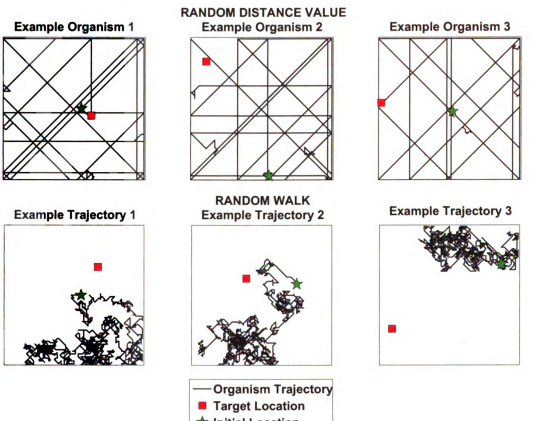


Figure 2.7: Sample trajectories for maximum sensor noise and random walk organism. Trajectories shown for the most abundant genotype at the end of evolution runs for the replicate populations with the three highest Best Distance Ratios for the random distance value treatment (top). Random walk trajectories were generated with the same hand-coded organism, injected into three different locations. All rotation uses the *tumble* instruction.

2.3 Conclusions

We demonstrated several notable results with the experiments on evolving tactic behavior in a simplified, idealized environment. First, we provided proof of concept for incorporating motility into the Avida system. Second, we highlighted interesting aspects of evolving this fundamental navigation behavior, including satisficing and robust performance in the presence of sensor noise. Perhaps the most interesting result was the evolution of the rudimentary memory mechanism that enabled the storage of the previously experienced gradient concentration for later use.

These results of these experiments underscore that memory is a critical hurdle to evolving intelligent behavior. The absence of an existing memory mechanism was clearly a stumbling block in evolving gradient following, in both noise-free and noisy treatments. In the experiments that follow, I explore the issue of evolving memory in more depth.

Chapter 3

Evolving Precursors to Memory: Reflex Behaviors

A reflexive response is an innate behavior that "has not been changed by learning processes" (Tinbergen, 1951, p. 2). Such behavior is unlikely to change due to an individual's experience. The response is hard-coded in the organism's genome, so the organism will always make the same behavioral choice when presented with the stimulus. A reflex action is typically identified as one that is involuntary and occurs immediately in response to the stimulus. For my experiments, the stimulus is a sensory cue from the environment. In general, we expect a reflexive response to evolve when (1) the environment is stable, *i.e.*, the cue is consistently associated with the same outcome for the organism, (2) the benefit of encoding the response outweighs the cost of making the encoding, and (3) the response has some selective advantage.

From a theoretical point of view, it is reasonable to think that reflexive, or "hard-wired" responses evolved before learning (Todd & Miller, 1990), and these types of responses are well known as the basis for conditioning (*e.g.*, Rescorla, 1988). In addition, we can imagine a scenario where it is better to *always* do a certain task than it is to *never* do that task (hence, the task should evolve); but it may be better still if the organism can selectively *choose* to not perform the task under some circumstances. For example, in a hostile environment, it is better to run away

from any larger organisms that come near you than to never run at all. But it is better still if you can identify the approaching organism and know when it is actually necessary to run.

This initial step of evolving a reflex response is necessary. From a practical standpoint, if an Avidian cannot evolve to perform an action correctly when it always should, it will never be able to effectively decide to take that action selectively.

3.1 The Avida State Grid

Inspired by the maze-learning experiments with bees, discussed in section 1.2.2, I designed a series of experiments to explore the evolution of memory in a navigation task. Similar to the bee maze experiments, Avidians evolved to follow a path in a virtual environment. My primary interest in this work was to study evolving memory use in individual organisms, independent of organism interactions. Since the context of the experiments includes movement, we implemented a new virtual environment, called the Avida state grid, as a technical solution to certain limitations of the existing Avida world. In the state grid, each organism has separate information about its environment. An individual organism remains in the same location in the Avida world throughout its lifetime, but also has a virtual grid where it can move. A single organism is the sole occupant of its state grid; in a sense, the state grid sets up a situation where the environment may be shared by different organisms, but each individual is oblivious to the existence of all other individuals, and its experiences are not influenced by the presence or actions of other organisms. The only way that organisms affect each other is that the offspring of a higher-fitness organism may replace a lower-fitness individual. The state grid provides several other benefits: larger populations can be used, without regard for allowing space for movement within the world, the simplified environment is easily defined and understood by human experimenters, and experiments run efficiently with relatively low computational overhead.

A single instance of a state grid is an $x \times$y toroidal grid that has a user-definable set of states, and each space on the grid has a particular state. Since the grid is implemented as a torus, an organism is, in effect, on an infinite plane: a movement trajectory that reaches a grid boundary wraps around to the opposite side of the grid, eliminating edge effects and minimizing the possibilities for organisms to exploit the geometry of the environment.

State grids may be defined with arbitrary x and y dimensions; there is no constraint, for example, that the state grid be square. The definition of the state grid includes the organism's inital (x, y) location on the grid, its initial facing, and the details of the grid itself. State grids are defined in the Avida environment configuration file, and each state grid is associated with a reaction that triggers the path traversal task.

3.1.1 State grid paths

I devised several different state grid environments, based on the experimental maze designs in Zhang *et al.* (2000). Each state grid contains a single "path" for the Avidians to follow. The idea behind the paths is that the organism must move around its environment to find sparsely distributed "food" resources. All movement requires energy, so each step depletes the organism's energy store. When an organism moving along the path encounters food, the food gives the organism more energy than the amount lost through the movement. Locations that are off the path contain no food, and so are "empty." When an organism moves into an empty location, the organism loses a small amount of energy, without regaining any energy. Numerous movements into empty locations are detrimental to the organism: continued energy depletion will impair the organism's ability to replicate, and increase the chance that the organism will be replaced by a higher-energy organism. Organisms that move along the food-rich path build up their energy, and are able to execute at an accelerated rate.

I varied the configurations of the paths according to the specific experimental design, using some combination of the following states:

1. **Nutrient:** A state that indicates the path, and contributes to building metabolic bonus (the "food" on the path).

2. **Directional cue:** A state that indicates a turn to either the right or left (a single 45-degree increment in the specified direction) is needed to remain on the path. Directional cues also contain a nutrient, and as such contribute to building metabolic bonus.

3. **Repeat-last:** A special directional cue to repeat the last direction of turn, and contributes to building metabolic bonus.

4. **Empty:** A state that indicates cells that are off of the path, and reduces the metabolic bonus. The net loss in energy from a step into an empty cell counteracts the gain of energy due to a step on the path.

3.1.2 Path traversal task

In these experiments, organisms received metabolic rate bonuses for a path traversal task. An organism's bonus depends on how well the organism negotiates the path. An organism that travels the entire path without stepping off the path receives the maximum possible bonus. Using the organism's state history, the task calculates the number of unique valid path steps, and subtracts the steps into empty states:

$$traversed = valid - empty \tag{3.1}$$

where *valid* is the count of unique path cells encountered, and *empty* is the total count of empty states encountered. I had both conceptual and practical reasons for counting movements to each path cell only once, but counting all movements into cells that are off the path. Conceptually, the path cells are analogous to food patches. The organism consumes most of the food in the patch the first time it moves into a path cell. Subsequent visits to a previously visited location supply only enough food to offset the energy lost in moving to the location. On the other hand, empty cells are always empty, and movement always requires energy. Each

step into an empty location results in a net loss of energy, because the organism cannot replenish its energy stores at that location. From a practical perspective, I counted the traversed path cells only once to encourage organisms to travel along the path, discouraging the evolution of less desirable behaviors. For example, if each step into a path cell were counted, organisms might evolve to oscillate between two path cells, never moving beyond those cells. Movement into empty cells should always be detrimental, since it should always be better for the organism to stay on the path than to move off the path.

I used the *traversed* value to calculate the *task quality* (TQ):

$$TQ = (traversed/pathLength) * processValue \tag{3.2}$$

with *traversed* as defined above in Equation 3.1, *pathLength* is the total count of path cells (nutrients and directional cues), and $processValue$ is a user-defined value that sets the bonus maximum.

I used the TQ value to determine the organism's metabolic rate bonus. I computed the metabolic rate (MR) as:

$$MR = \begin{cases} (BaseRate) * 2^0 & \text{if } traversed <= 0 \\ (BaseRate) * 2^{TQ} & \text{if } traversed > 0 \end{cases} \tag{3.3}$$

This approach delivers an exponential reward, doubling the organism's metabolic rate bonus for each step on the path that is not counteracted by a step off the path into an empty cell. I chose this strategy to encourage organisms to continue traveling on the paths. This reward system sets up a situation where it is always better to move than to remain still, and it is better yet to continue moving on the path.

3.1.3 State grid instructions

We implemented four state grid instructions that provide organisms with movement and sensing capabilities on the state grid. The *sg-move* instruction moves the organ-

Table 3.1: State values for state grid experiments. The definition of a state grid includes a listing of the state of each cell in the grid. The *sg-sense* instruction returns the value shown for each state.

State	Return Value
Empty	−1
Right turn	2
Left turn	4
Repeat last turn	1
Nutrient	0

Table 3.2: NOP-modified behavior of *if-grt-X* and *if-equ-X* instructions.

NOP Label	Value for Comparison
nop-A	−1
nop-B	2
nop-C	4
Default (no NOP)	1

ism into the cell it is currently facing. Organisms can change their facing by using the *sg-rotate-r* and *sg-rotate-l* instructions, which change the organism's orientation by one 45-degree increment to the right or left, respectively.

The *sg-sense* instruction returns a user-defined value that represents the state in the organism's current cell. These state values are provided in the state grid definition; Table 3.1 lists the state values that I used for these experiments. The return values were not used directly for any calculations, *e.g.*, task quality. I chose these particular values because they can be manipulated with relative ease using the assembly-code-like Avida instruction set. The values were consistent across all state grids in this and subsequent experiments. This consistency simplified the organisms' problem of mapping a particular action to a particular state. In such an environment, I expected successful organisms to evolve to differentiate between the states and take specific actions based on the different states.

I also added two new comparison instructions, *if-greater-than-X* (*if-grt-X*) and *if-equal-to-X* (*if-equ-X*), that supplemented existing comparison instructions. These instructions allow an organism to compare the value in its BX register to a predefined value. A *no-op* (NOP) label immediately following the comparison instruction

Table 3.3: Instruction set used for experiments. Instructions shown in italics are not part of the default instruction set. See also Appendix A.1 for explanation of instructions.

Instruction	Instruction	Instruction
nop-A	inc	jmp-head
nop-B	dec	get-head
nop-C	add	if-label
if-n-equ	sub	set-flow
if-less	nand	*if-grt-0*
pop	IO	*sg-move*
push	h-alloc	*sg-rotate-l*
swap-stk	h-divide	*sg-rotate-r*
swap	h-copy	*sg-sense*
shift-r	h-search	*if-grt-X*
shift-l	mov-head	*if-equ-X*

determines the value to use in the comparison, as shown in Table 3.2. I added the new comparison instructions because an Avida organism has to combine several different arithmetic instructions in order to compare a register value to any specific value. The new *if-equ-X* and *if-grt-X* instructions provided a shortcut and simplified the comparison process for the Avidians, and also contributed to evolved genomes that were simpler to analyze. The details of these new instructions do not adversely affect the adequacy of our model, since my focus in the experiments is on memory; the mechanisms of constructing comparisons are not relevant to my questions of interest. It is also important to note that early pilot experiments that did not use the new comparisons successfully evolved the desired behavior. Table 3.3 shows the complete instruction set used for all state grid experiments.

3.2 Experiments

3.2.1 State Grids and Experiment Design

Within the Avida state grid setup, as described in Section 3.1, I implemented two types of environments, each containing a single "path" for the Avidians to follow.

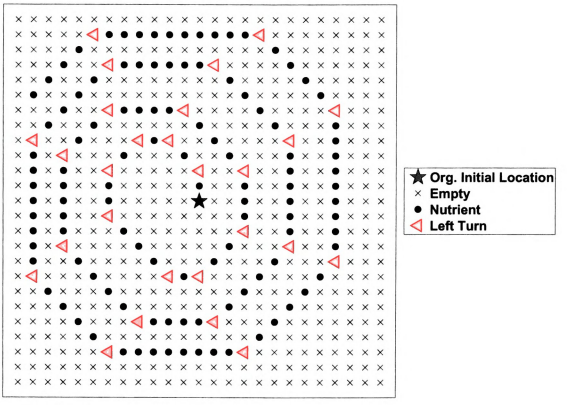


Figure 3.1: Example state grid "path." This grid illustrates a single-turn, left-turn-only environment. Blue circles indicate the "nutrient" state, yellow triangles show cells with the "signpost" state, **x** appears in each cell with the "empty" state, and the organism's initial location is shown by the green star.

One environment type contained turns in a single direction (*i.e.*, one path instance contained only right turns, while another path instance had only left turns, as illustrated in Figure 3.1). All paths used only 45-degree turns, so that a direction change could be reliably accomplished with a single, unmodified Avida instruction. The single-direction paths had a spiral shape, and contained three states: directional cue (right or left, but only one type per grid), nutrient, and empty. The other environment type contained four states, and paths had both right and left turns. For each environment type, there were four different paths that organisms were exposed to over the course of evolution. An individual organism experienced only one state grid path in its lifetime, but its ancestors and descendants experienced different paths during the course of evolution. I used different paths in order to discourage the evolution of algorithms that were brute-force solutions of a particular path instance.

In the first suite of experiments, I set up an environment that provided certain regularities. I put a directional cue at each turn on the path. This strategy gave organisms the information they needed to make the turn decision, at the point in time and space that the decision was made. An organism could sense the cue by using the *sg-sense* instruction; each state (nutrient, right turn signpost, left turn signpost, empty) had a different return value (see Table 3.1). The organism had to decide what, if anything, to do with the information from the cue.

For the first group of experiments, I used four different single-direction state grids, two that contained only right turns, and two that had only left turns. The second group of experiments used five state grids that contained both right and left turns. For both sets of experiments, organisms were assigned one of the state grids at random upon birth and replication. In this way, different generations of organisms experienced all the possible environments through the course of evolution. For each environment type (*i.e.*, single-direction paths and right-left paths), 50 replicate experiments ran for 250,000 updates, with a population maximum of 3,600 organisms (60×60 world). I seeded all runs with the default simple self-replicator ancestor (*i.e.*, an organism that has only the capability to replicate), and used the default mutation rates (0.085 genomic mutation rate for a length 100 organism, comprising a 0.0075 per site copy mutation rate, 0.05 per divide insertion mutation rate, and 0.05 per divide deletion mutation rate (Ofria & Wilke, 2004)).

3.2.2 Results and Discussion

Single-direction Paths

To evaluate the success of different evolved populations, I used both quantitative performance measures and behavioral tests of evolved organisms. For the quantitative measures of performance, I examined fitness and task quality over time. These values are tracked and recorded during the course of an Avida experiment. The behavioral tests involved running execution traces of evolved organisms on different state grid configurations. The trace record includes the trajectory history of the

organism, given in the form of state grid cell identification numbers that can be converted into state grid (x, y) coordinates for visualization.

Fitness and task quality provide information about how well the organism performs in a given environment. In Avida, fitness equates to metabolic rate divided by the number of instructions required for an organism to replicate. For these experiments, I tied the metabolic rate bonus exclusively to the path traversal task. Task quality measures the ratio of the total possible metabolic rate bonus an organism will receive, based on its performance of the tasks in the environment, and so serves as a straightforward measure of an organism's performance of a given task. For example, an organism with a task quality of 0.03 is relatively low performing in the related task, while an organism with a task quality of 0.98 is relatively high-performing, receiving a higher metabolic rate bonus.

In the state grid experiments, task quality is a direct measure of how much of the path an organism traversed without moving into any empty cells (see section 3.1.2 for the definition of the path traversal task): an organism that traversed all paths without moving into any empty squares would have a task quality of 1.0. Because overall metabolic rate (and therefore fitness) for these experiments was associated solely with the path traversal task, task quality and fitness track closely. To illustrate this point, Figure 3.2 shows both the average $log(fitness)$ of the final dominants (prevalent genotype in a population) and the average maximum task quality (AMTQ) from all 50 replicate experiments for the single-direction path state grids. The similarity of the curves shows that task quality and fitness are similar in this particular situation, and that task quality can be used as the performance metric for the experiments. Figure 3.2 also illustrates that the best-performing organisms from the 50 replicate experiments navigated, on average, half of each path experienced at the end of 250,000 updates of evolution. Figure 3.3 shows the distribution of AMTQ values for these same experiments, with each path shown separately. Although the sample medians appear lower for the left-turn paths, there was no significant difference between the AMTQ distributions for each path, as measured by the AMTQ at the end of evolution (Kruskal-Wallis Test, $p = 0.287$). Qualitatively, the overall

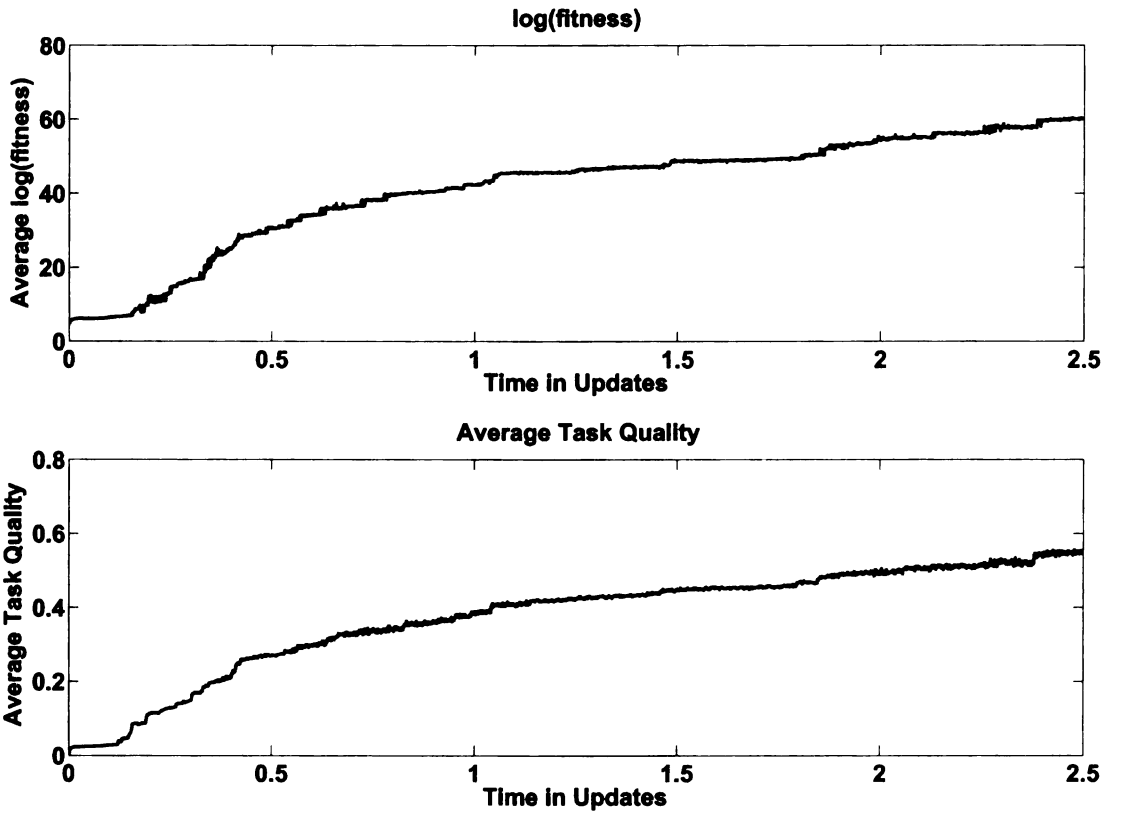


Figure 3.2: Average $log(fitness)$ and average maximum task quality (AMTQ) in single-direction paths experiments. In this case, $log(fitness)$ and task quality track each other closely, and are both good measures of algorithm performance. The curves also show that the average task quality over all 50 final dominant organisms is near 0.6, indicating that, on average, the organisms successfully traversed over half of each path by the end of the evolution run.

performance appears somewhat noisier on the left-turn-only paths. This characteristic is also evident in the performance of the population average task quality and AMTQ of the top-performing population from the set of 50 replicate experiments (Figure 3.4).

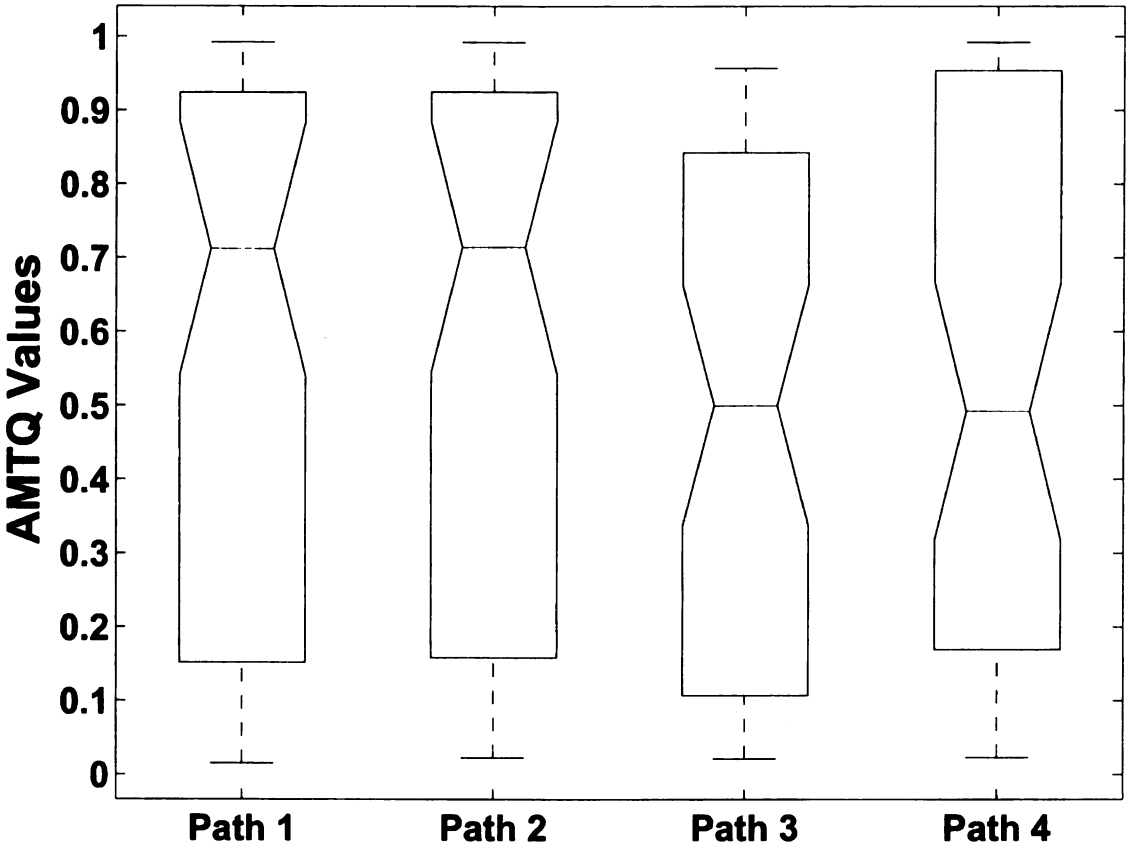


Figure 3.3: Distribution of average maximum task quality (AMTQ), individual single-direction paths. Paths 1 and 2 are right-turn-only paths, Paths 3 and 4 are left-turn-only paths. Although the medians for right-turn and left-turn paths appear different, there is no significant difference in the AMTQ distributions (Mann-Whitney U-test, $p > 0.05$ for all pairs of paths; see Table ?? for specific p values).

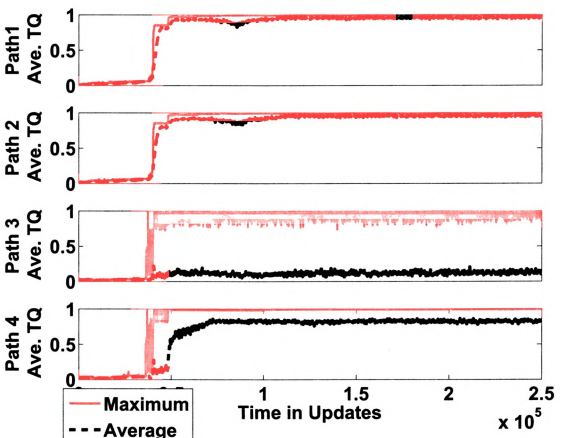
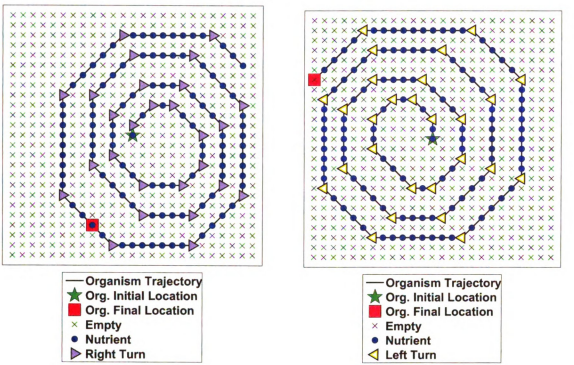


Figure 3.4: Average maximum task quality (AMTQ) and population average from the case study population in single-direction path experiments. The plot shows the AMTQ for each path separately.

To test the behavior of evolved organisms, I ran execution traces for selected final dominants in different environments. With the first environment, I tested organisms on the same state grids that the organisms experienced during evolution. My goal in these tests was to observe how the organisms' strategies function in the environments where the behavior evolved. In the second environment, I exposed the organisms to novel environments, *i.e.*, state grids and paths that the organisms had not experienced during evolution. I used the novel paths in order to demonstrate the generality of the evolved solutions, or uncover solutions that had been tuned specifically to the evolutionary environments. In this technique, I systematically removed some of the regularities from the paths. With such changes in the paths, an organism navigating on a novel path encounters situations that evolution never had to deal with, thus probing the organism's underlying algorithm. For example, I presented organisms with new configurations of single-direction paths. These new paths had the same fundamental characteristics as the paths in the ancestral environments (*i.e.*, each path contains only right turns or left turns), but the details of the paths varied (path length, number of turns, steps between turns, state grid size). An organism with a brute-force solution to the paths of the native environments would be unable to follow the new path, since its algorithm will depend on the path regularities that were present during evolution. On the other hand, an organism with a more generalized strategy for path-following will navigate a new single-direction path with ease. For a more challenging test, I also exposed the organisms from the single-direction path ancestral environment to new environments that have both right and left turns in the same path. These organisms never encountered this situation during evolution in their ancestral environment. If an organism is able to succeed in the novel environment, with no additional evolution, the organism must have some mechanism for dynamically switching between turn directions, even though this circumstance never occurred during the organism's evolutionary history. I used this technique of placing organisms in novel environments for each set of state grid experiments.

Figure 3.5 shows trajectories of the final dominant with the highest ending metabolic rate among all 50 replicate single-direction path experiments, on a right-



(a) Right-turn Path (b) Left-turn Path

Figure 3.5: Trajectories of an example evolved organism on paths that were experienced during evolution.

turn-only path (Figure 3.5a) and on a left-turn-only path (Figure 3.5b). This organism was also the final dominant in one of the top ten populations ranked by AMTQ. The organism's trajectories on the other two evolutionary environment state grids are qualitatively identical to those shown, and so are not included here. The organism's evolved strategy performed well in both turn environments. The organism did some "backtracking" on the right-turn grid, *i.e.*, it turned around and moved back in the direction of its initial position, retracing its steps on the path. This fact did not result in a reduction of the organism's fitness: although the task quality increases only for unique path cells encountered, the calculation does not penalize an organism for multiple traversals of a path cell. The only potential danger for an organism that is "killing time," like this one, is that the organism could be copied over by another organism that manages to replicate first. This particular organism not only avoided that fate, it also evolved to be the most fit individual in its population. This organism was able to navigate the entire right-turn path, and

did not enter any empty cells when moving on the right-turn path. The organism also successfully followed the left-turn-only path, stopping after it encountered a single empty cell. This particular organism was not able to successfully negotiate paths containing both right and left turns. On four of the five paths of this type, the organism failed to replicate, when it became "stuck" at the first turn cue that indicated a change of direction (*i.e.*, from right to left or from left to right). When traversing the fifth path, the organism failed to recognize a turn cue, moved into an empty cell, and then completed replication.

Analysis of the execution trace of this organism as it traverses each of these two paths reveals how its algorithm operates to produce this behavior. Most of the path-following and replication code of this organism's genome is organized into two modules. The first module, "Module A," is mostly concerned with moving on a right-turn, while the second module, "Module B," focuses on left-turn paths and also contains a nested copy loop. These code sections are both executed, regardless of whether the organism is on a right-turn or left-turn path, but the behavior that the modules produce differs according to the path type (*i.e.*, right or left). In general, Module A is a "counting" routine. When the organism is on a right-turn path, Module A counts the organism's steps. On a left-turn path, Module A counts the number of rotations the organism executes. Module B allows an organism moving on a left-turn path to travel to the end of the path, and then replicate. When the organism is on a right-turn path, the organism uses Module B to "backtrack" on the path, retracing some of its steps, while it finishes its replication process.

The following is a pseudocode description of the functionality of Module A:

```
do
  rotate right
  if (CX > 0) copy
  copy
  CX <- sense
  if (CX == nutrient) rotate left
  else if (CX == right turn)
    CX <- 128
    move
  BX <- BX + 1
while (BX != CX)
```

This code executes differently, depending on whether the organism is currently in a right-turn or left-turn environment. When traversing a right-turn grid, the organism uses this loop to count its steps to the end of the path. Setting the CX register to the value of 128 (by reading the current position of the Instruction Pointer (IP)) and incrementing the value in the BX register (which begins at a value of 0 at the first loop iteration) with every loop iteration sets up the exit condition for the loop: after the organism has taken 127 steps in the loop, the last increment of the BX register causes execution to exit the loop. When executing this loop on a left-turn-only path, the organism remains in the same spot and executes the loop four times, performing a one-eighth turn in each iteration. When the value of the BX counter reaches 4, the organism exits the loop, and is now facing in the "wrong" direction (*i.e.*, facing back the way it has already come). The section of code immediately following this module includes another four one-eighth turns, so the organism regains the facing it had upon entering Module A.

Module B operates as follows:

```
do
  move
  BX <- sense
  if (BX != nutrient) rotate left
  if (BX == empty)
    while (!end label) copy
  else if (!end label) copy
  if (end label) divide
while BX != empty and not end label
```

When this algorithm is executed on a left-turn-only path, the organism moves along the path, eventually moving one step off the end of the path into an empty cell. At that point, the organism "stands still," and executes a tight copy loop to complete copying its genome to its offspring, at which time it divides. On a right-turn path, however, the organism never enters the tight copy loop; instead, it copies just one instruction for each iteration of Module B, while it retraces its steps along the path. This strategy produces the backtracking that is visible in the trajectory plot of Figure 3.5a. The number of instructions needed to produce an offspring remain similar on right- and left- turn paths (1779 instructions for the right-turn path shown in Figure 3.5a and 1780 instructions for the left-turn path shown in Figure 3.5b) since an extra instruction is copied with every iteration of Module A when the organism is moving on a right-turn path. Table 3.4 lists the Avida instructions for the two modules described above. I also give a more detailed discussion of the organism's step-by-step operation in Appendix A.2.1.

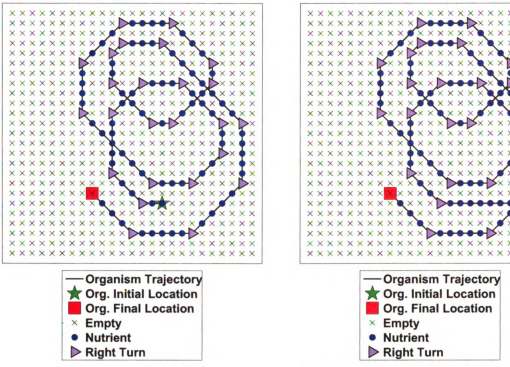
This organism's performance on *novel* paths was, however, inconsistent. The organism successfully negotiated a right-turn-only path, stopping after it encountered one empty cell at the end of the path (Figure 3.6a). On a novel left-turn-only path, however, the organism stopped as soon as it reached the first left-turn signpost (Figure 3.7a). The execution traces of the organism's traversal of these paths reveal the reasons for this performance difference. When following a left-turn path, the

Table 3.4: Avida instructions for example single-direction path organism.

Module A	Module B
sg-rotate-r	sg-move
if-grt-0	sg-sense
nop-C	nop-B
h-copy	if-n-equ
h-copy	sg-rotate-l
sg-sense	if-equ-X
nop-C	pop
jmp-head	if-less
sg-rotate-l	h-search
if-equ-X	if-label
get-head	nop-C
sg-move	h-divide
inc	h-copy
if-n-equ	mov-head
mov-head	

organism "expects" the first turn to be encountered within the first two or three movements. When this does not occur, the organism's execution enters an infinite loop (Module B discussed above) when it reaches the first turn, and the organism cannot replicate. This problem does not hamper traversal of a novel right-hand path. In addition, the organism does not backtrack along the path as it did on the right-turn paths that were experienced during evolution: the organism stops after one step into an empty cell, successfully finishes the replication loop, and divides. These results suggest that the organism's algorithm is reasonably well generalized for right-turn environments, but not for left-turn environments. The organism's performance on a novel left-turn path does improve if the distance to the first turn is shortened to a few steps.

To test this idea, I altered the organism's initial position on the paths shown in Figure 3.6a and Figure 3.7a. The initial position of the right-turn grid was moved farther away from the first turn, and the initial position for the left-turn grid was moved closer to the first turn. Execution traces for the organism on the two grids support the preceding analysis. The organism is still able to successfully traverse



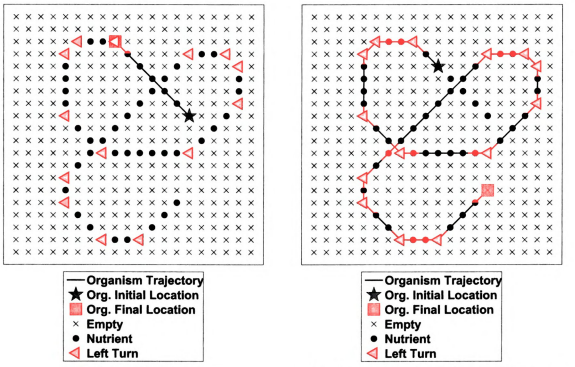
(a) Right-turn Novel Path

(b) Right-turn Novel Path, New Initial Position

Figure 3.6: Trajectories of an example evolved organism (same organism as shown in Figure 3.5) on novel right-turn path. Figure (a) shows the organism's trajectory from the original initial position for the path, and Figure (b) shows the trajectory from a new initial position that created an extended distance to the first turn on the path (7 steps added to the distance to the first turn).

the right-turn grid, showing that the distance to the first turn does not matter in a right-turn path. With fewer steps to the first turn on the left-turn path, the organism is now able to navigate the left-turn path as well. Figures 3.6b and 3.7b show the organism's trajectories on the novel paths with new initial locations.

A model by Stephens (1991) demonstrated that within-generation persistence of environmental factors is more important than between generation persistence as a selective force in the evolution of learning. For the single-direction path experiments, the single-direction characteristic of each path furnishes the within-generation regularity. Changeable features include the details of the path pattern and the path length. These experiments also present between-generation regularity: the offspring has a 0.25 probability of having an environment identical to its parent's environ-



(a) Left-turn Novel Path

(b) Left-turn Novel Path, New Initial Position

Figure 3.7: Trajectories of evolved organism (same organism as shown in Figure 3.5) on novel left-turn path. Figure (a) shows the organism's trajectory starting at the path's original initial position, and Figure (b) shows the trajectory from a new initial position that shortens the distance to first turn (unused nutrient locations from original path appear for comparison).

ment (*i.e.*, the offspring may be given the same state grid as its parent), and a 0.50 probability of having the same environment type as its parent (the offspring may be given a path with the same turn direction as its parent's path). Even if the offspring is born into an environment with the opposite turn direction to its parent's environment, the core of the path-following problem has its own inherent regularities.

Given all these predictable factors, it would seem that the most advantageous strategy for this situation should be to evolve some static solution to the problem. The preceding discussion, however, points toward some level of plasticity to the evolved solution used in the case study. This plasticity may be at either the genomic level, or at the execution level. In other words, the plasticity may involve *static execution flow*, where differences in the environment produce different outcomes

Table 3.5: Number of genomic positions required for path following and the resulting dynamic plasticity ratios (DPR) for the final dominants from the replicate experiments with the highest average maximum task quality (AMTQ) in single-direction path experiments. Three replicates tied for the top tank (*i.e.*, their AMTQ values were the same), and two tied for the next rank. The rank 9 organism evolved as a one-direction "specialist" *i.e.*, it could function well only on right-turn paths), and so was not considered in the DPR measurements.

Rank	Replicate ID	Sites Both Directions	Sites Left Only	Sites Right Only	Plasticity Ratio
1 (tie)	8	55	1	4	0.0847
1 (tie)	14	60	6	7	0.1780
1 (tie)	48	91	13	6	0.1727
4 (tie)	9	58	3	6	0.1343
4 (tie)	31	78	1	2	0.0370
6	32	49	1	3	0.0769
7	19	50	5	12	0.2537
8	30	47	0	3	0.0600
10	46	40	7	14	0.3444

using the same instructions, or *dynamic execution flow*, where different instructions are executed, causing different actions to be chosen, depending on the environment (Clune, Ofria, & Pennock, 2007). As a measure of the proportion of dynamic *versus* static plasticity, I computed a *dynamic plasticity ratio* (DPR). This ratio examines several components of an organism's genome: 1) the number of sites (instructions) that are important on both right-turn and left-turn paths; 2) the number of sites important only for right-turn paths, and 3) the number of sites important only for left-turn paths. The total of these three components identifies the number of sites that are important to the path-following task. The ratio of the number of sites used for only one direction to the total sites for path following is the dynamic plasticity ratio, which quantifies the execution-level plasticity of a genome:

$$DPR = \frac{Right + Left}{Right + Left + Both} \tag{3.4}$$

where $Right$ is the count of sites in the genome that are important only for right-turn paths, $Left$ is the count of sites important only for left-turn paths, and $Both$

is the count of sites needed for paths in either direction.

Existing Avida analysis tools identify sites used in specific tasks, through the use of knock-out experiments. I analyzed the results of the task mapping analysis by evaluating whether knocking out a particular site decreased the organism's metabolic rate for each condition of interest (*i.e.*, right-turn path, left-turn path, both path directions). That analysis gave site counts for *Right*, *Left*, and *Both* to produce the DPR.

I ran these analyses on a hand-coded organism that was able to complete the path traversal task, in order to provide a basis for comparison of DPR values. This organism's DPR value was 0.074 (25 genomic positions for both turn directions, 1 position for right, 1 position for left). This DPR value shows little dynamic plasticity in the hand-coded organism.

I also performed the analyses on the final dominants from the replicates with the ten highest average maximum task quality (AMTQ) values from the single-direction paths evolution runs (*i.e.*, the ten replicate populations with the highest AMTQ at the end of the evolution runs). Of those ten organisms, one had evolved as a right-turn "specialist" (*i.e.*, it was able to negotiate the right-turn paths but not the left-turn paths), and so was not included in the DPR analysis. Table 3.5 summarizes the site counts and DPR information of the evolved genomes.

The range of DPR values is surprising, ranging from a minimum of 0.0370 to a maximum of 0.3444. The "step-counter" organism, discussed above in detail, has high DPR, 0.2537, while the dominant from the population with the highest AMTQ has a low DPR, 0.0847. Such a wide range of DPR values among the dominants of the top-performing populations suggests that, even in a highly regular environment where a static solution can perform well, some level of execution-level plasticity may offer an advantage. Given the nature of this environment, some minimal level of plasticity is almost a requirement for high performance: organisms must, at the least, have a mechanism for turning both directions. Specialists, like the right-turn specialist organism described above, can do well by executing one type of path perfectly, but a generalist with a good solution for both directions will do better.

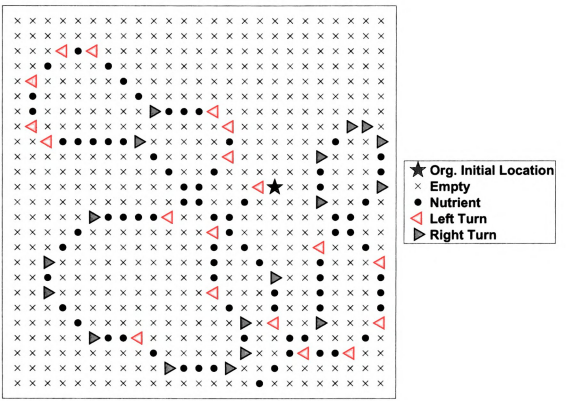


Figure 3.8: Example right-left turn path.

Right-left Turn Paths

The right-left turn environments present fundamentally the same problem to evolution as the single-direction environments. The right-left turn environments are somewhat more complex, however, since evolution must always contend with both turn directions in every path.

The state grids for the right-left turn experiments were set up in the same manner described in section 3.2.1, using four states: nutrient, right signpost, left signpost, and empty. Figure 3.8 shows an example right-left turn path. Five different paths were used for this set of evolution experiments. All other experiment details are identical to the single-direction path experiments, as described in section 3.2.1.

The performance at the end of 250,000 updates of evolution across all 50 replicate experiments, as measured by the average maximum task quality (AMTQ), was not significantly different from the populations in the single-direction path experiments

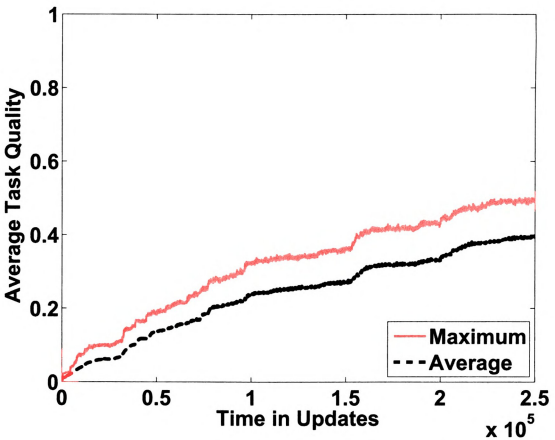


Figure 3.9: Average maximum task quality (AMTQ) and population average task quality for all 50 replicate populations in right-left turn path experiments. Task quality shown is averaged over all five paths experienced during evolution.

(Mann-Whitney U-test, $p > 0.05$). In the current experiments, the AMTQ at the end of the evolution experiment was close to 0.5, as opposed to 0.6 in the single-direction path experiments (Figures 3.9, 3.10). This difference appears to support the intuition that evolving a solution for paths that contain both directions is slightly more challenging than the single-direction path. There was, however, no significant difference in the performance on each path, measured by the AMTQ at the end of evolution (Kruskal-Wallis Test, $p = 0.950$). Some populations, however, evolved good solutions. The final dominant of the top-performing population demonstrated near perfect performance early in evolution, as shown by its task quality of close to 1.0 on all paths at time 50,000 updates (Figure 3.11).

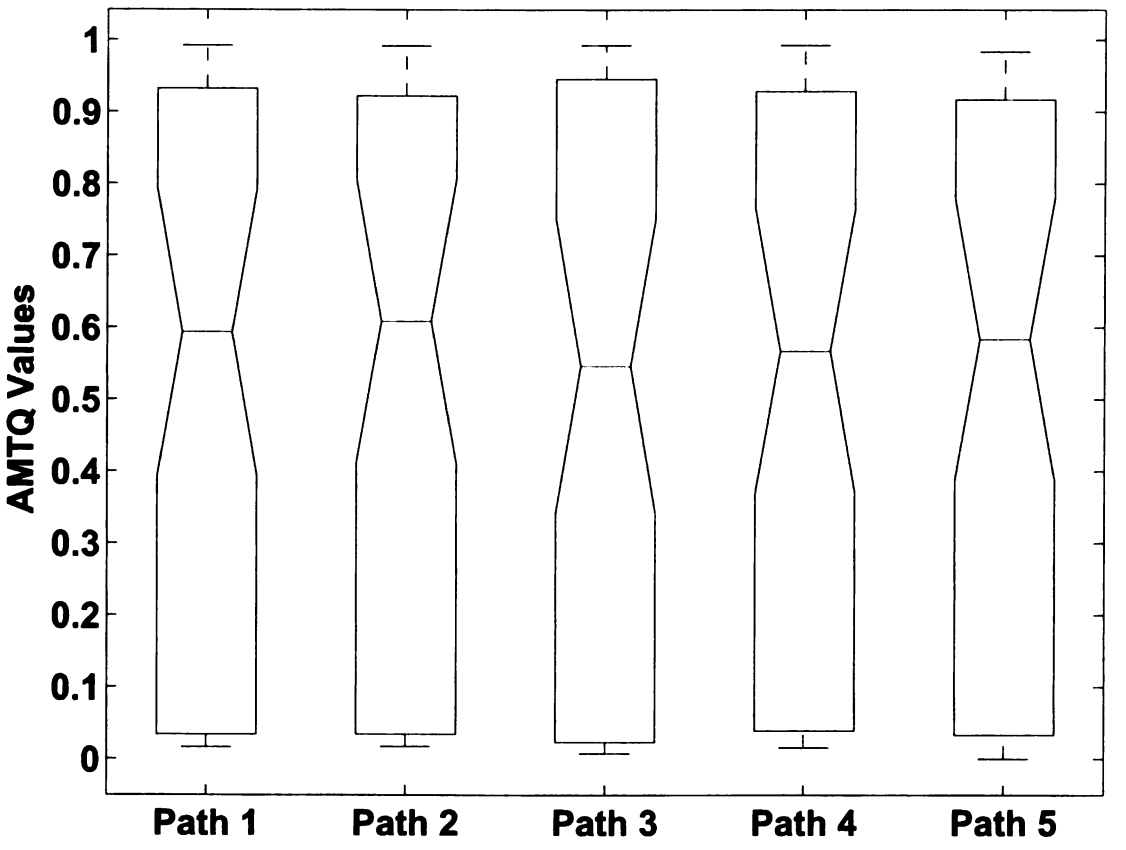


Figure 3.10: Distribution of average maximum task quality, right-left turn experiments, individual paths. There is no significant difference in the AMTQ distributions (Kruskal-Wallis Test, $p = 0.950$).

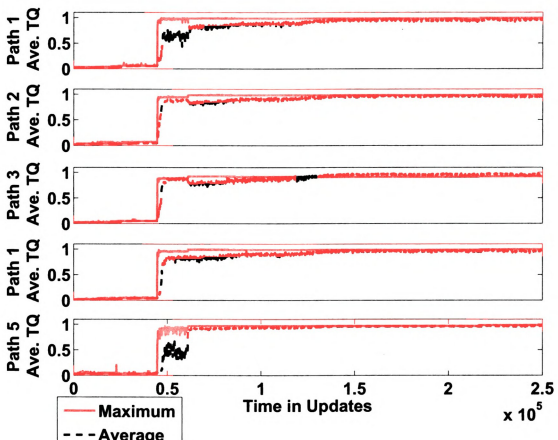
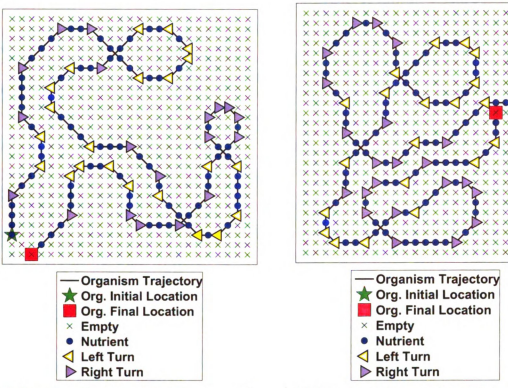


Figure 3.11: Average maximum task quality (AMTQ) and population task quality average, top population in right-left turn path experiments. Task quality is shown for each individual path experienced during evolution.



(a) Sample trajectory, right-left turn evolutionary path 1.

(b) Sample trajectory, right-left turn evolutionary path 2.

Figure 3.12: Trajectories of example evolved organism from right-left turn experiments.

I again tested the behavior of evolved strategies by running execution traces of the highest AMTQ final dominant among all 50 replicate experiments as the organism navigated both known paths (*i.e.*paths that had been experienced during evolution) and novel paths. Figure 3.12 shows trajectories of this top performing organism in two of the five evolutionary environments. The organism traverses each path in its entirety, stopping after taking one step off the end of the path into an empty cell. This organism performs equally well on the other paths experienced during evolution, so those trajectory plots are not shown here.

The organism also successfully negotiated novel paths. Figure 3.13 shows trajectories of the highest AMTQ final dominant organism, traced on two different novel paths. The dimensions of the grid containing both of these paths are different from the dimensions of the grids in the evolutionary environments: the novel paths shown have dimensions of 20×20 and 19×17 (Figure 3.13 (a) and 3.13 (b), respectively),

as opposed to the 25×25 grids that were experienced during evolution. Since all state grids are toroidal, the grid dimensions should make no difference to organisms, and organisms do not have access to any global information. However, I included tests with these different grid dimensions to provide additional evidence that the evolved algorithms do not work by finding and exploiting geometrical information, such as grid size, but instead function through gathering and using information from the environment.

This organism executes most of its movement with a concentrated movement loop. At a high level, the general structure of the code is *move-sense-decide*. The decision concerns whether or not to turn, and if a turn is to be made, which direction to turn. Within the loop, conditional statements guard the turn directions to provide the correct execution flow for each environmental cue. The organism has a simple, but clever, mechanism for using the default behavior of the comparison instructions to select the correct action, based on the current sense information. The organism can use the default comparison behavior because it manipulates the current sensed state value such that the values match the comparisons as needed. I provide more detail of how this mechanism works, after the following pseudocode description.

In pseudocode, this movement loop functions as follows:

```
do
  if (BX > 1) rotate left
  copy
  move
  BX <- sense
  BX <- right-shift(BX)       # See details below
  if (BX == 1) rotate right  # Executes if last sense == Right
  else if (BX < CX)           # Skip when sense == Left
    if (BX > 0) continue      # Iterate if sense != Empty
while (BX > 0)
```

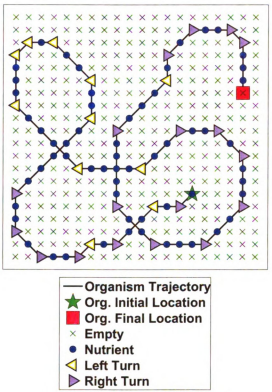
The key detail of this loop's execution is how the right-shift operation prepares the sensed state value for use with the unmodified comparison statements. Stepping

Table 3.6: Avida instructions for example right-left turn path organism.

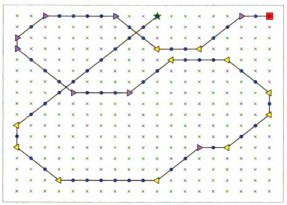
Movement Loop
if-grt-X
sg-rotate-l
h-copy
sg-move
sg-sense
shift-r
if-equ-X
sg-rotate-r
if-less
if-grt-0
mov-head

through the algorithm, starting from the **BX<−sense** line, the current cell state is sensed, and the value placed in BX. That value is then right-shifted, dividing most of the sense values by 2. Recall the state grid return values from Table 3.1. If the current state is nutrient (return value = 0), BX is still 0; if the state is right-turn (return value = 2), BX is now 1; if the state is left-turn (return value = 4), BX is now 2; if the state is empty (return = -1), BX is still -1 (since the operation is an arithmetic right-shift, the sign bit is preserved in the shift). This low-level manipulation of the sense value permits the algorithm to use the default behavior of the comparison instructions, thus avoiding the need for NOP modification of the instructions.

This particular solution is simple and economical, accomplishing the job with few extraneous instructions. The organism has evolved an equally economical copy loop near the end of its genome. The copy loop performs the bulk of the organism's replication, and begins execution only after the movement loop has terminated. Table 3.6 gives the Avida code for the example organism's movement loop, and I provide more detailed analysis of how the organism's code operates in Appendix A.2.2.



(a) Right-left Turn Novel Path 1



(b) Right-left Turn Novel Path 2

Figure 3.13: Trajectories of example evolved organism from right-left turn experiments on novel paths containing both right and left turns. Figure (a) shows the organism's trajectory on a path in a 20×20 grid, while Figure (b) illustrates the trajectory on a 19×17 grid path. The dimensions of both of these grids differ from the dimensions of the evolutionary environments (*i.e.*, 25×25).

3.3 Conclusions

In this chapter, I focused on evolving reflex actions in response to environmental stimuli. This was an important step in the experimental design, from both theoretical and practical perspectives. From a theoretical standpoint, reflexes were likely the evolutionary precursors of more flexible responses, and are well-studied as bases for conditioning. On the practical side, we cannot hope to evolve more complex faculties such as memory and learning in Avida if we cannot evolve reflexive behaviors. These experiments started at a basic level, using simple environments. The single-direction turn experiments presented the simplest environments, where the only variation between the environments was in the direction of turns and the details of the environment layout. The second phase, the right-left turn paths, extended the single-direction path environments to include both right and left turns in each environment.

Interesting and clever strategies evolved for both environment types, despite the simplicity and regularity of those environments. The evolved strategies were not rote recapitulations of known patterns in the evolutionary environments: organisms were able to follow paths that they had never experienced during evolution. The evolution of the "step-counter" organism was a particular highlight of these experiments. Mechanisms that are analogous to step counters play important roles in animal odometry, and investigating how such mechanisms evolve is of great interest.

Both change and regularity in environments have roles to play in promoting the evolution of memory and learning (Stephens, 1991). The environments used in the experiments in this chapter contain many regularities, but incorporate enough change to suggest that some amount of execution-level plasticity may be useful. To test this idea, I quantified the execution-level plasticity of selected genomes with the dynamic plasticity ratio (DPR). The DPR values varied widely, and did not have a clear story to tell about the desirability of such plasticity in the experimental environment. However, the presence of higher DPR values suggests that dynamic plasticity may provide some benefit even when static solutions can suffice.

Chapter 4

Evolving One-bit Memory

Environmental complexity can present organisms with problems, opportunities, or a mixture of both. Complexity can be described as "variety, diversity, doing many different things, or having the capacity to occupy many different states" (Godfrey-Smith, 2002, p. 232). Complex environments are demanding, requiring organisms to cope with irregularity and ambiguity. Successful behavior may necessitate decisions that incorporate past actions as well as the current situation. In complex environments, memory and learning will often provide needed tools for increasingly flexible responses.

Flexible behavior spans a wide range of responses. Basic types of reversible plasticity fall in a part of that range, and may be considered as early forms of learning; learning itself is a complex and advanced form of plasticity (Dukas, 1998) (see also section 1.2.1). This connection between basic reversible plasticity and learning leads to the next step in investigating the evolutionary pathways for memory and learning: evolving a plastic response. Clune *et al.* (2007) demonstrated that plastic responses can evolve in Avida. Their experiments resulted in successful strategies that used either static flow of execution (the same set of instructions are executed in the same order, but different environmental inputs result in different behaviors) or dynamic execution flow (different segments of instructions are executed depending on the current environment). Dynamic execution flow emerged in some cases in the experiments discussed in Chapter 3.2.2. For the experiments discussed

in this chapter, I focused on two questions:

1. How does the use of memory change when evolution occurs in more complex environments?

2. What are the differences between evolving life-long, stable memory *versus* short-term, volatile memory?

4.1 Evolving One-bit Lifetime Memory

In the next set of experiments I investigated evolving a life-long memory, specifically to produce individuals that store experience that guide their future actions. This relates to the idea of phenotypic plasticity, discussed above and earlier in section 1.2.1, where a given genotype may express differently in an individual organism in response to environmental differences. Here, we looked for different execution of the Avidian's genome, depending on the specific environment the organism found itself in.

4.1.1 Experiments

The fundamental design of these experiments is the same as the preceding single-direction path experiments, described in section 3.2. The important difference between the experiments relates to the cues I used to construct the paths. In the experiments of the last chapter, I put a directional cue at each turn; a right turn and a left turn have different return values (see section 3.1.3 for more details). In that setup, past cues never had to be stored in order to make an informed decision about the current action. In this second set of experiments, there is a slight twist on that theme. In these "cue-once" environments, only the first turn on the path gives a specific directional cue; all subsequent turns have a different sense cue that, in effect, signals an organism to "repeat the last turn direction." This "repeat-last" sensory cue is consistent, both within a single state grid and across all state grids, *i.e.*, it is the same value whether the current path is a left-turn path or a right-turn path.

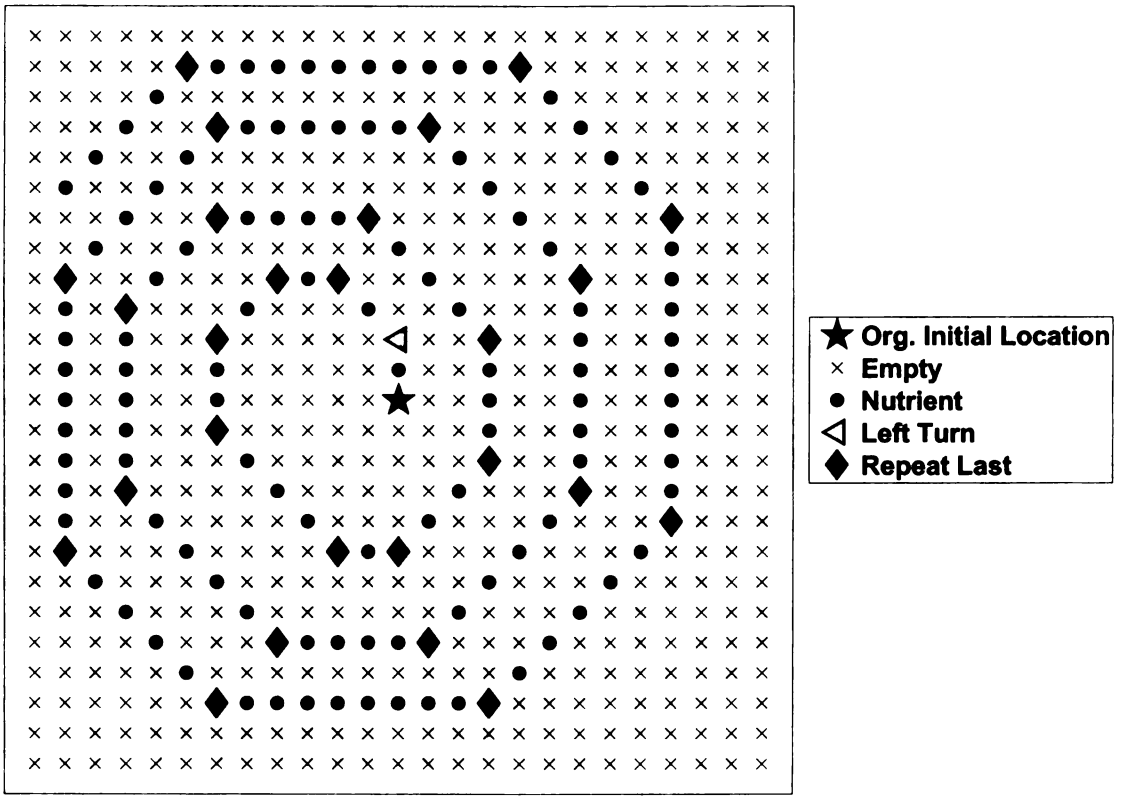


Figure 4.1: Example state grid from "cue-once" experiments. The state grids for this experiments have the "cue-once" directional cue and "repeat-last" states. The one directional cue, shown by the yellow triangle, signals whether the environment is a right-turn or left-turn environment (here, a left-turn environment). Subsequent turns (brown diamonds) are signaled by the "repeat-last" state, that gives the same return value in both right and left turn environments, serving as a cue to "repeat the last turn."

The repeat-last cue is therefore consistent within an environment, but ambiguous between environments. The idea behind the cue-once setup is that organisms will need to evolve to be able to identify the environment type from the one specific directional cue, and then be able to maintain that directional information throughout its lifetime. The capacity to store that information is analogous to a "life-long" memory, meaning a memory that needs to be maintained and used, but does not have to be changed after it is stored. This also resembles phenotypic plasticity, as discussed earlier: a particular genotype may express differently (in the case of an Avidian, this means it will execute differently) in response to different environmental conditions. This type of dynamic execution-flow plasticity (Clune et al., 2007)

may be either reversible or permanent: if reversible, an organism will be able to function flexibly in differing environment types during its lifetime; if the plasticity is permanent, an organism will be incapable of performing appropriately in another environment once execution for one environment begins.

As in the earlier single-direction path experiments, an organism was randomly assigned one of four path grids (from a set of two right-turn paths and two left-turn paths) at birth. The paths were the same as those used in the preceding single-direction path experiments, with the states of all turns but the first turn changed to the repeat-last cue, as described above. Figure 4.1 illustrates a path from the cue-once paths of the current experiments. The other details of the experimental setup and reward structure were identical to those of the experiments presented in Chapter 3.

4.1.2 Results and Discussion

Once again, I used task quality to assess performance over the course of evolution. The average task quality for all four environments (maximum and population average for all 50 replicate experiments) is shown in Figure 4.2. The curves of both the maximum and the population average appear to still be gradually rising at the end of the 250,000 update evolution run, suggesting that continued evolution may have resulted in additional performance improvement. The population with the highest AMTQ also appears to be making some improvements at the end of the run (Figure 4.3). Figure 4.4 shows the distribution of AMTQ values for each individual path. The average maximum task quality (AMTQ) values at 250,000 updates in this experiment were not significantly different from those measured in the single-direction path experiments (Mann-Whitney U-Test, $p = 0.759$) and in the right-left turn experiments (Mann-Whitney U-Test, $p = 0.546$). The lack of significant difference between the average task quality indicates that evolution was able to discover effective strategies for the somewhat more challenging environment presented in the cue-once experiments. There was also no significant difference in the performance on each path, measured by the AMTQ at the end of evolution

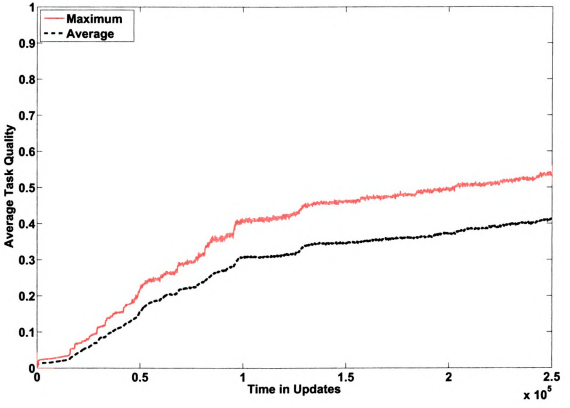


Figure 4.2: Average maximum task quality (AMTQ) and population average, all 50 replicate populations in cue-once experiments. Task quality shown is averaged over all four paths experienced during evolution.

(Kruskal-Wallis Test, $p = 0.805$). These evolved strategies enabled the organisms to navigate large portions of the cue-once paths, indicating that evolution succeeded in building algorithms that differentiate the information from the first turn cue, and use that information to guide the rest of the execution along the path.

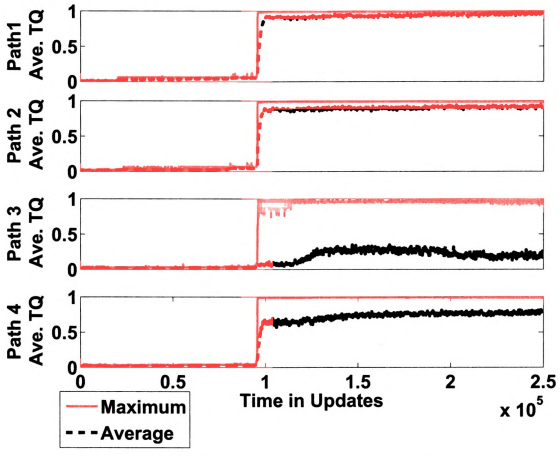


Figure 4.3: Average maximum task quality (AMTQ) and population average, top performing population in cue-once path experiments, paths shown separately.

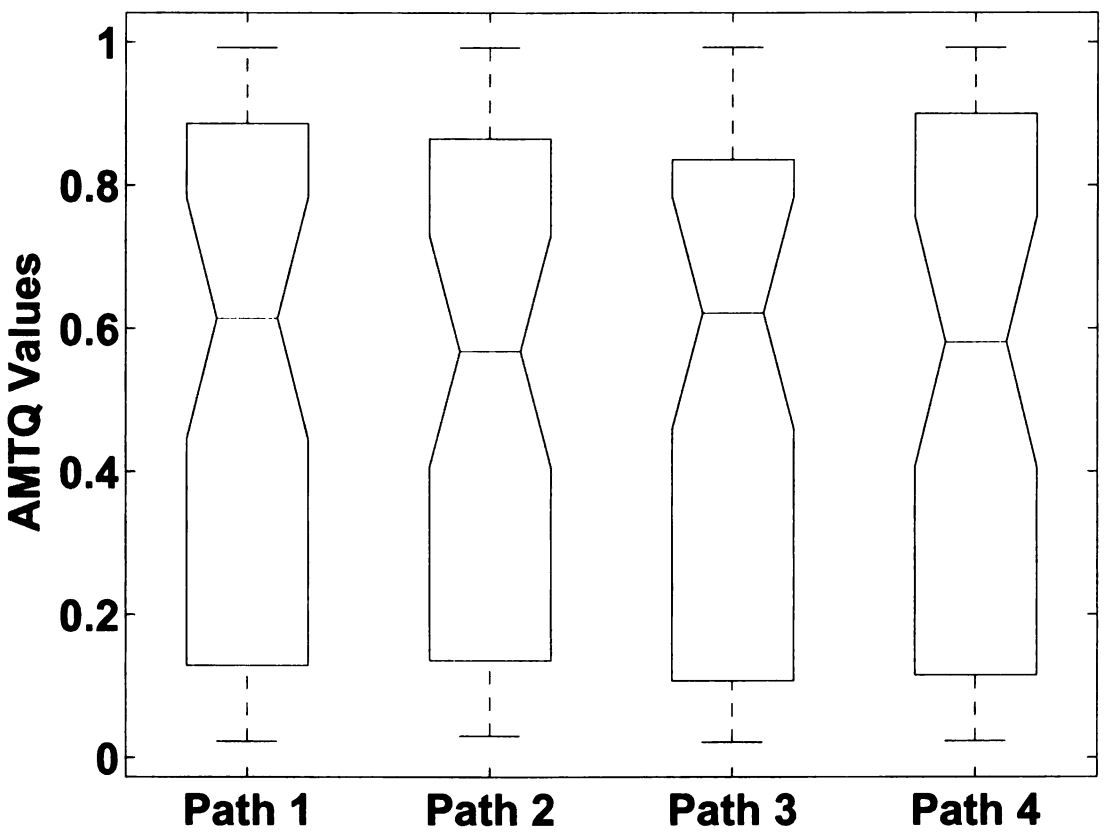
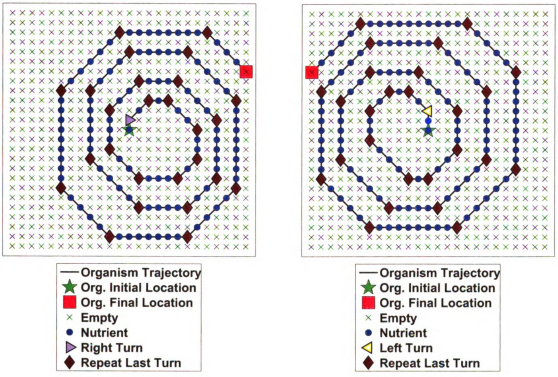


Figure 4.4: Distribution of average task quality values, individual cue-once paths. Paths 1 and 2 are right-turn only paths, Paths 3 and 4 are left-turn only paths. There is no significant difference in the AMTQ distributions (Kruskal-Wallis Test, $p = 0.805$).



(a) Right-turn Path (b) Left-turn Path

Figure 4.5: Trajectories of example evolved organism on paths that were experienced during evolution in cue-once experiments.

Figure 4.5 shows trajectories of the final dominant organism from the population with the highest average maximum task quality. Analysis of an execution trace revealed that the genome has two clear movement modules, one used primarily for moving on right-turn paths, "Module A," and the other. "Module B," used exclusively for movement on a left-turn path. The code for Module A comes before Module B in the genome, and is executed for both right-and left-turn paths. For left-turn paths, this loop is used to move on the straight portion of the path that precedes the first turn. An organism moving on a right-turn path divides shortly after exiting Module A, so Module B is executed only when the organism is moving on a left-turn path. Module B also contains a nested copy loop, which the organism executes once it steps off the end of the path into an empty cell.

A pseudocode version of Module A of this genome is as follows:

```
do
  if (BX != CX) rotate right  # Skip when last sense==nutrient
  move
  copy
  BX <- sense
  if (BX > 1) copy          # True when sense==right or left turn
  if (BX > 1) BX <- BX -1 # True when sense==right or left turn
  if (BX != CX)
    if(BX == 1) continue  # Loop if sense==right turn, nutrient
while (BX != empty OR BX != left turn) # Exit if sense==empty, left
```

This code produces different behavior depending on the path type (right or left) the organism is traveling on. On right-turn paths, the right turn executes only if the organism last sensed a right-turn cue; the turn is skipped if a nutrient was sensed. The *copy* instruction always executes, and when the organism is on a right-turn path, it uses this module to perform most of its replication. Execution remains in this module until the organism senses an empty cell (*i.e.*, it steps off the end of the path). When the organism senses an empty cell this loop terminates, and the organism divides when it encounters a *divide* instruction that is positioned ten

instructions after the end of this module. When the organism is traversing a left-turn path, the right-turn instruction never executes, so the organism continues straight on the path. Execution exits this module when a left-turn cue is sensed; the instruction immediately following loop exit is a left turn (*sg-rotate-l*), which executes. Execution then moves on to Module B, after executing a small section of instructions that contain no path-following instructions.

Pseudocode for Module B shows its underlying logic:

```
while ( )  # No exit condition for loop; divide terminates loop.
  rotate left
  if (BX == 1) move # Step after a left turn
  if (BX > 0)
    if (BX < CX) rotate right
  move
  BX <- sense
  if (BX < CX)              # True if last sense==empty
    copy                    # Enter copy loop only if sense==empty
    if (end label) divide
  copy                      # Copy while still moving on path
```

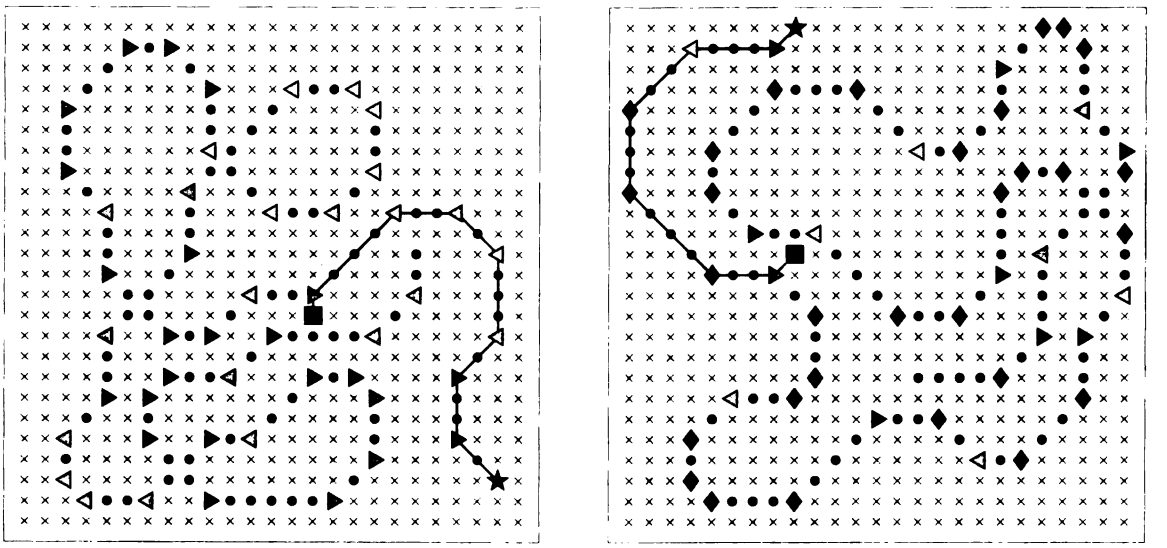
As previously noted, this module executes only when the organism is moving on a left-turn path. A left turn is executed immediately after loop entry, and is followed by a step. Taking a step without sensing first works in the ancestral environments because there are no cases of turns on two consecutive steps on the path. The next two comparisons work together to ensure that the organism is facing correctly, depending on whether a repeat-last or a nutrient is sensed. The comparison **if (BX > 0)** is true when the repeat-last cue is sensed; **if (BX < CX)** is executed, but is false, so the right turn does not execute. The comparison **if (BX > 0)** is false when a nutrient cue is sensed, so the next comparison is skipped, and the right turn executes. This negates the left turn that was done at the top of the loop, and the organism moves straight forward on the path. Execution enters the copy loop only

Table 4.1: Avida instructions for example cue-once path organism.

Module A	Module B
if-n-equ	sg-rotate-l
sg-rotate-r	if-equ-X
sg-move	sg-move
h-copy	if-grt-0
sg-sense	if-less
if-grt-X	sg-rotate-r
h-copy	sg-move
if-grt-X	sg-sense
dec	swap-stk
if-n-equ	if-less
if-equ-X	h-search
mov-head	h-copy
	if-label
	nop-C
	h-divide
	mov-head

when the organism senses an empty cell, and the organism remains in the copy loop until it finishes copying and divides. Table 4.1 shows Avida code sections for this organism's two movement modules. I give more a detailed analysis of the operation of the organism's Avida code in Appendix A.2.3.

Given the structure of this genome, the question is whether an organism can "switch" readily between directions. Since the right-turn segment of the genome is always executed, an organism may be able to transition into left turns after doing some right turns. However, the converse (right turns after left turns) appears impossible, since there is no mechanism in the execution for returning to the earlier part of the genome. To test this, I traced trajectories of the organism as it executed on two novel paths, one from the earlier right-left turn paths, and another that incorporated right and left cues and repeat-last cues (the second path type will be used during evolution in the next phase of experiments). Figure 4.6 shows the trajectories. The trajectories demonstrate that the organism is able to transition from right turns to left turns, but it cannot switch to right turns after performing left turns. In this



(a) Novel Path, Right and Left States

(b) Novel Path, All Turn States

Figure 4.6: Trajectories of example evolved organism from cue-once experiments, tested on novel paths: (a) Trajectory on a path using the right/left turn states at every turn, as in the earlier right-left turn path experiments; (b) Trajectory on a path that incorporates right, left, and repeat-last states.

genome, the change to left-turning is irreversible. The left-turn loop of this genome is a clear example of a part of a genome that executes in one environment but not in another environment.

4.2 Evolving One-bit Volatile Memory

In constructing his model of the evolution of animal learning, Stephens (1991) considered the tension between the importance of environmental change, on the one hand, and environmental regularity, on the other hand. Stephens terms the case for the importance of change the *absolute fixity argument*: assuming that there are some costs to learning, an absolutely fixed environment should produce a genetically fixed behavior pattern rather than learned behavior. In other words, in a fixed environment, there is nothing to learn. The other viewpoint, which Stephens calls the *absolute unpredictability argument*, contends that if the state of today's environment has no predictive relationship to the state of tomorrow's environment, there is no point in learning. In a random or chaotic environment, there is, once again,

nothing to learn. Stephens' model resolves this paradox by taking into account two distinct types of environmental regularity ("within-generation persistence" and "between-generation persistence"), and argues that the evolutionary value of learning is determined by "the pattern of predictability in relation to an animal's life history" (Stephens, 1991, p. 77). In other words, it is necessary to consider not only the overall environmental predictability, but also the relationship between predictability and an organism's life history. The model shows that within-generation persistence (the extent to which today's state predicts tomorrow's state within an individual's lifetime) is more important as a selective pressure in evolving learning than between-generation persistence (the extent to which states in the parental environment predict states in the offspring's environment): when there is some amount of change, increased within-generation persistence promotes learning, while increased between-generation persistence appears to have no effect. However, if an environment is nearly completely fixed, increasing change either between or within generations promotes the evolution of learning.

The details of Stephens' model present an environment and population that differ from those in the current experiments (*e.g.*, non-overlapping generations, an environment containing exactly two resources), but the ideas of the model are clearly pertinent to the present study. In the environments of the state grid experiments, there is a high degree of both between- and within-generation persistence. The parental environment is a good predictor of the offspring's environment; in general, there is a 0.25 probability that an offspring's environment will be identical to that of its parent, and the environment never changes dramatically. The environment is temporally fixed but spatially heterogeneous; the primary unpredictability is what the next state might be.

Even with such simple environments, we can reasonably expect some learning to evolve. In this case, the learning is quite fundamental, involving remembering one bit of information. The "cue-once" experiments of the preceding section (section 4.1.1) took a step in that direction. Those experiments focused on evolving "life-long memory": once the bit was stored, it never had to be changed within an organism's

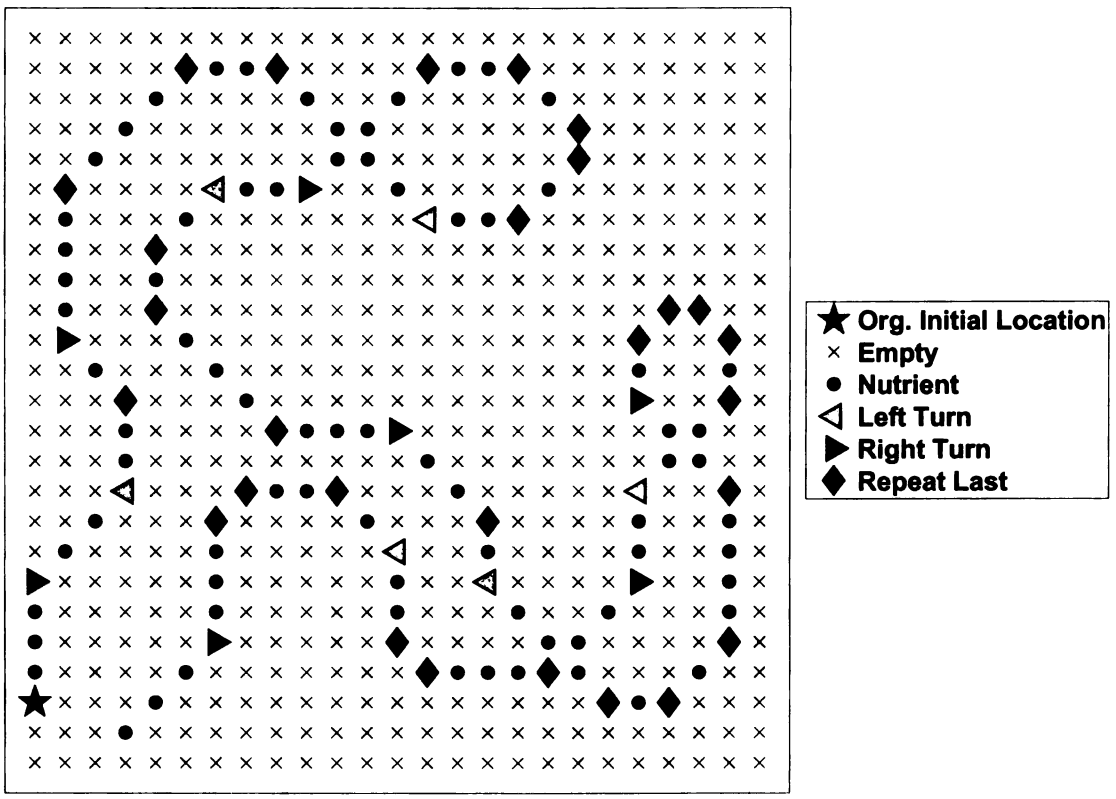


Figure 4.7: Example state grid with irregular path. A "new" turn direction (the first turn or a change from the previous turn direction) is cued by the unique right or left turn sense value. Subsequent turns in that same direction (brown diamonds) are signaled by the "repeat-last" state, that gives the same return value when preceded by either a right or left turn, cueing to "repeat the last turn."

lifetime. In the final set of experiments, I deal with evolving more active one-bit memory, or one-bit volatile memory. For these experiments, evolution needed to contend with intermittent updating of information.

4.2.1 Experiments

The general layout of the state grid paths for these experiments used the right-left turn paths from the experiments in section 3.2.2, but changed some of the cue return values. For the irregular paths in these experiments, I used the same state structure as the cue-once paths, except that each path contained both right and left turns. If a turn is encountered as a "new" direction—that is, either as the first turn, or a change from the previous turn direction (*e.g.*, a left turn after one or more right

turns)—the sense value is the directional cue value. Otherwise, if the current turn is the same direction as the preceding turn, the sense value is "repeat-last," as seen in the cue-once experiments. This arrangement of information along the path means that an Avidian must change the "remembered" sense cue value an unknown number of times in its lifetime, and at irregular intervals. Such an arrangement necessitates flexible use of information from an increasingly complex environment.

Figure 4.7 shows an example irregular path environment. These state grids use all the cues introduced in the other experiments: nutrient, right turn, left turn, repeat-last, and empty. The specific sense values for all cues were the same as those used in all other experiments, as summarized in section 3.1.1. Other details of the experimental setup remain the same as in all previous experiments.

4.2.2 Results and Discussion

As in the previous experiments, I used the average maximum task quality (AMTQ) as the primary performance metric for evaluating the evolved strategies. The overall AMTQ, shown in Figure 4.8, shows a markedly weaker performance than in the other three experimental environments. The difference in AMTQ at the end of 250,000 updates was significantly different in the irregular path experiments compared to the other three environments (Mann-Whitney U-test, $p < 0.05$ in all pairwise comparisons between the current experimental treatment and the other three experimental treatments). There was, however, no significant difference in the performance on each path, measured by the AMTQ at the end of evolution, as shown in Figure 4.9 (Kruskal-Wallis Test, $p = 0.238$). This environment may simply be more difficult than those of the previous experiments. The AMTQ of a single experiment replicate, shown in Figure 4.10, illustrates some of the effects of the underlying evolutionary dynamics, and a degree of fragility in the evolving algorithms at many points during the course of evolution, as mutations perturbed the functioning of the algorithm.

Despite the generally inferior performance of the evolved populations in this environment, some highly effective strategies evolved. Figure 4.11 illustrates the trajectories of the final dominant organism from the population with the highest

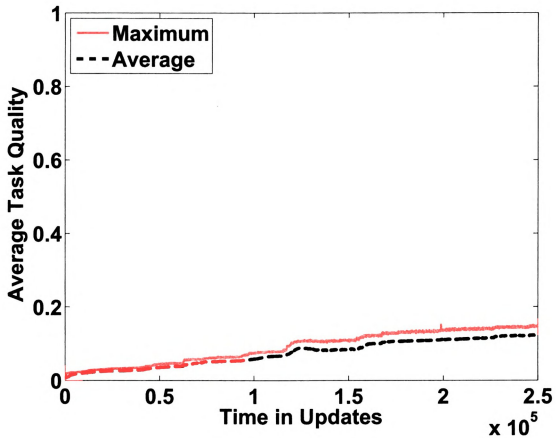


Figure 4.8: Average maximum task quality (AMTQ), irregular paths. The plot shows the combined AMTQ and the overall population average task quality for all four paths experienced during evolution.

AMTQ at the end of the 250,000 update evolution run. This organism has an excellent solution for following these paths, stopping after taking one step off the end of the path into an empty cell. The evolved algorithm is equally effective on novel paths, as shown in Figure 4.12.

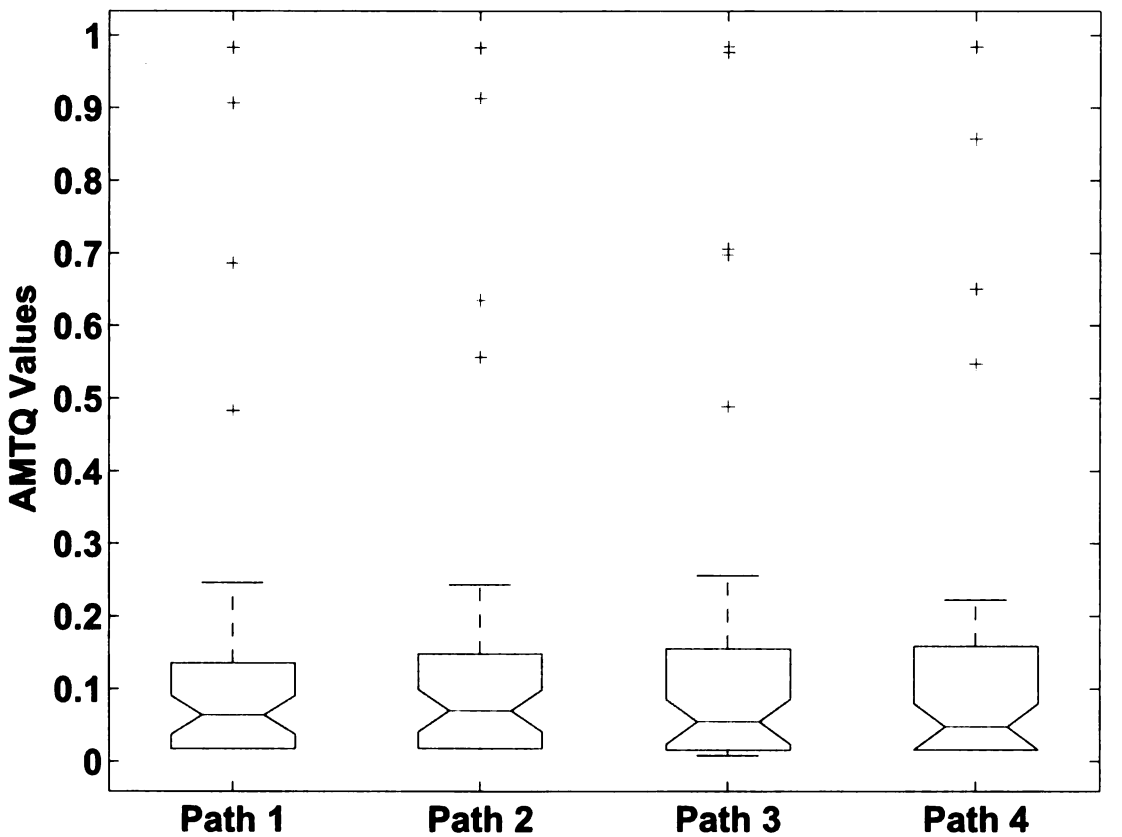


Figure 4.9: Distribution of average maximum task quality (AMTQ)values, individual irregular paths. There is no significant difference between the AMTQ distributions (Kruskal-Wallis Test, $p = 0.238$).

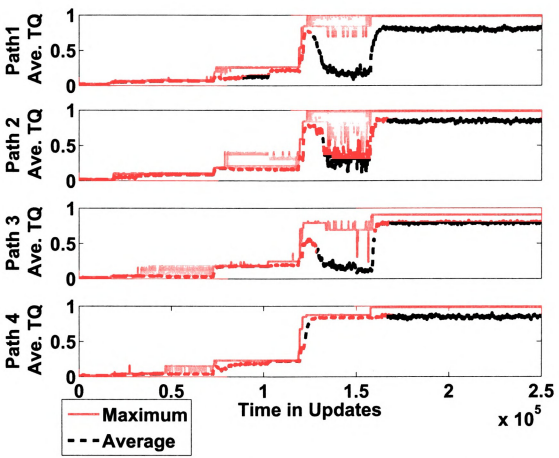
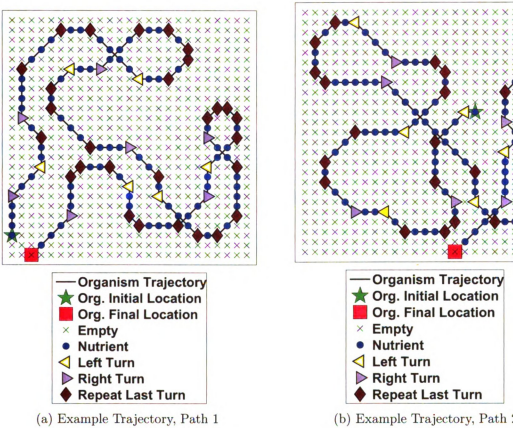


Figure 4.10: Average maximum task quality (AMTQ) for a single replicate in irregular paths environment. The plots shows the combined AMTQ and the population average task quality for each individual path experienced during evolution. This population had the highest AMTQ at the end of the 250,000 updates of evolution.



(a) Example Trajectory, Path 1

(b) Example Trajectory, Path 2

Figure 4.11: Trajectories of evolved organism, irregular path experiments. Trajectories of example organism on paths that were experienced during evolution in irregular path experiments. In both (a) and (b), the organism stops after encountering one empty cell.

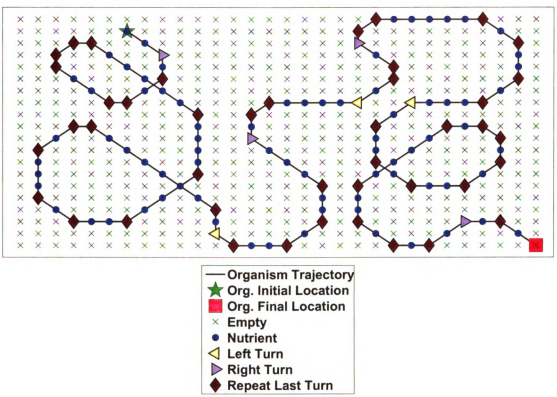


Figure 4.12: Example trajectory, example evolved organism from irregular path experiments on a novel path.

The execution of this organism's genome is somewhat complicated, and shows an impressive degree of flexibility. In general, this organism operates by moving its execution to different parts of its genome based on the sensed environmental state. The organism accomplishes all of its path-following with two loops, one for moving through left-turn path sections, "Module A," and the other for moving through right-turn path segments, "Module B." Unlike the other organisms that I have examined in detail, this organism has well-defined functional and structural modularity for handling right-turn and left-turn path sections. Module A appears before Module B in the organism's genome. Module A can perform an arbitrary number of consecutive left turns, and any number of forward steps. Module A has a nested loop that produces straight-ahead movement on the path; execution remains in the smaller loop unless a non-zero cue is sensed. Sensing a left-turn or repeat-last cue triggers an exit of the smaller loop, but execution remains in Module A. Sensing a right-turn or empty cue results in exiting the smaller loop, and will also cause execution to exit Module A. There is a section of instructions between the modules that has no *sg-move* instructions, but does have a single right-turn instruction that negates the last left turn performed before exiting Module A. Using Module B, the organism can maneuver through right-turn path sections. Module B functions with arbitrary numbers of forward steps and repeated right turns. If a left turn cue is sensed, Module B terminates and execution jumps to the beginning of the genome, eventually reaching Module A again. If an empty cell is sensed while execution is in Module B, the module terminates and execution continues with the instructions after the module. In addition to the movement modules, the organism has a tight copy loop near the end of its genome that accomplishes almost all the copying for the organism's replication.

Module A, for navigating on left-turn path sections, functions as follows:

```
do
  do
    move
    BX <- sense
    if (BX < CX) swap (BX, CX) # Executes if sense==empty
    BX <- BX - 1
  while (BX < CX) # Exit for any state but nutrient
  rotate left
while (BX == repeat last) OR (BX < CX)# Exit for sense==right, empty
```

The decrement of the value in BX following the sense instruction manipulates the value in the BX register such that execution remains in the nested loop as long as the organism is sensing nutrient (meaning that the organism is moving straight on the path), but will exit the nested loop when any other cue is sensed. Whenever this module is executing, the value in CX is 0 at the top of the loop. Executing **BX <− BX−1** with the nutrient return value (0) places a value of -1 in BX, so execution does not exit the nested loop. Decrementing the repeat-last return value (1) places a value of 0 in BX, causing execution to exit the nested loop and do the left turn. When the right-turn return value (2) is decremented, the value in BX becomes 1, and the nested loop is exited. Execution then exits Module A, after executing the left turn. The swap of values in BX and CX is executed only if an empty cell is sensed. The swap places 0 in BX, and -1 in CX, so the nested loop is exited, and execution leaves Module A after the left turn, since the continuation condition fails (now BX is equal to CX after BX is decremented).

Module B, for moving through right-turn path segments, is:

```
do
  rotate right
  if (BX < CX) BX <- sense
  move
  BX <- sense
  if (BX == repeat last) continue # Loop if sense==repeat last
  else if (BX == left turn) jump IP to 0 # Jump to start of genome
  BX <- BX + 1
  rotate left
while (BX != CX)
```

As noted earlier, an additional *sg-rotate-right* instruction executes before Module B entry and ensures proper orientation for turning right, since Module B contains both *sg-rotate-r* and *sg-rotate-l* instructions that always execute. Correct orientation is maintained by selectively executing the left turn at the end of the module. When a repeat-last cue is sensed, execution in Module B skips the left turn (since BX=1), and returns directly to the top of the loop. When a nutrient is sensed, BX=0, so the increment of BX and the left turn are executed. When the organism senses a left-turn cue, execution jumps out of Module B and returns to the beginning of the genome. As in Module A, the value in CX is 0 during execution of Module B. If an empty cell is sensed, incrementing the value places a value of 0 in BX, and execution exits the module. Once the organism moves into an empty cell, execution moves on to the copy loop near the end of the genome, and the organism completes its replication. Table 4.2 lists the Avida code for this organism's path-following modules, and I provide analysis of the Avida code in Appendix A.2.4.

There are two features of this organism that are particularly interesting. The first is the organization of the genome. The sections of the genome that do the bulk of the "work" for this organism—the two movement modules and the copy loop—are functionally and spatially modular. For all three of these loops, very little happens

Table 4.2: Avida instructions for example irregular path organism.

Module A	Module B
sg-move	sg-rotate-r
sg-sense	if-label
sub	nop-B
if-less	add
swap	nop-B
h-divide	if-less
dec	sg-sense
if-less	sub
mov-head	sg-move
push	nand
if-label	sg-sense
nop-C	if-equ-X
shift-r	mov-head
nop-A	
sg-rotate-l	
if-equ-X	
if-less	
mov-head	

within them apart from the main function of the loop. The loops are also spatially modular: they are located in different sections of the genome. Example organisms from the preceding experiments also demonstrate structural modularity, but their functional modularity is, generally, less defined. The second feature of special interest is the flexibility of execution flow between code modules. The execution flow enables the organism to cleverly handle all the contingencies of the environment. For example, even though Module A (left-turn module) is encountered first in the sequential execution of the genome, if a right turn is encountered first, the flow moves easily through Module A and into Module B (right-turn module). The algorithm evolved to deftly maneuver along the paths, using the information of the states in the environment to alter its exeuction.

4.3 Evolving Complexity

Compared to a natural environment, the current study's most complex environment is strikingly simple and sparse. Yet, the simple capabilities that have evolved in the preceding experiments—for example, sensing environmental information, discrimination between different sensory values, differential behavior based on environmental conditions, directed movement through the environment—interact to produce more complex behavior. Even these simple environments foster the evolution of complex features. There is a large body of evidence from biology that supports Darwin's ideas about the incremental evolution of complex organismal features, such as the eye. The Avida system allows for detailed study of the evolution of complexity that is difficult to accomplish in the natural world, due to factors such as long time spans, extinct intermediate organisms, and incomplete knowledge of biological mechanisms.

In their landmark study, Lenski *et al.* (2003) demonstrated the depth of insight that Avida experiments can provide to help shed light on the evolution of complex features and many other issues in evolutionary biology. The results of the study showed that complex functions (in this case, the ability to perform the most complex logic task, *equals* (EQU)) evolved by building on simpler functions that had evolved earlier. The experiments demonstrate "the validity of the hypothesis, first articulated by Darwin...that complex features generally evolve by modifying existing structures and functions" (Lenski et al., 2003, p. 143).

As a first step in investigating the evolution of complex features in the context of memory and simple navigation, I performed experiments that probed whether the evolution of a more complex capability was facilitated by a simpler capability that arose earlier in evolution. In other words, did simpler functions serve as building blocks of more complex functions? In the context of the current experiments, the increase in the task complexity is a mirror of the increasing complexity of the experimental environments. As the environments become more and more complicated, evolution must combine more and more simpler functions to successfully cope with the environments. Key among these functions is the ability to store and reuse in-

Table 4.3: Summary of ancestors and environments for transplant experiments.

Ancestral Environment	Transplant Environment
Single-direction paths	Right-left turn paths
Right-left turn paths	Cue-once paths
Cue-once paths	Irregular paths

formation from past experience, *i.e.*, memory. In addition to the general question related to the evolution of complexity, the more specific question of these experiments is: Does having the memory capability afforded by evolving in one environment provide an advantage when the environment and the accompanying memory demands change?

4.3.1 Experiments

For these experiments, I transplanted organisms that evolved in one experimental environment into the next, more complex environment. To do this, I selected ten final dominant organisms from each of the first three experiments (single-direction paths, right-left turn paths, cue-once paths) and seeded them into new runs of experiments in the next experimental phase. The organisms selected for transplanting had the ten highest average maximum task quality (AMTQ) values from the 50 replicate populations of each experimental setup. A high AMTQ value indicated an organism that performed well on all the paths it experienced (*i.e.*, completed much of the path length). Table 4.3 summarizes the ancestral environments and the related transplant environments.

Each of the ten ancestors seeded ten replicate populations for a total of 100 trials. Experiments began with a full population, by injecting the ancestor organism into all world grid cells (grid size 60×60, population maximum size of 3,600 organisms). The experiments ran for 250,000 updates, using the default Avida mutation rates. The experiments used the same sets of state grids that were used in the previous experiments, as well as the strategy of assigning each organism a randomly selected state grid at birth.

Table 4.4: Transplant experiments, ancestral and evolved task quality. Summary of ancestral average maximum task quality (AMTQ) and evolved AMTQ for surviving organisms in transplant experiments. Ancestral AMTQ was measured at the end of 3 generations with no mutations occurring. Evolved AMTQ was taken at the end of 250,000 updates of evolution. Ancestral AMTQ entries that appear as "—" indicate that the organism was not able to survive in the no-mutations test environment; the corresponding Evolved AMTQ values show that these organisms were able to adapt and survive after mutations in later evolution.

Ancestral Environment	**New Environment**	**Seed Organism Replicate ID**	**Ancestral AMTQ**	**Evolved AMTQ**
Single-direction	Right-left Turn	8	0	0.6264
		9	0.0155	0.9824
		19	—	0.9853
		29	0	0.8741
		30	—	0.9441
		31	0.0131	0.9441
		32	0.9802	0.9823
		46	0	0.5773
Right-left Turn	Cue-once	1	0.013	0.4222
		18	0.2405	0.8983
		25	0.2405	0.7535
		30	0	0.7700
		31	0.0131	0.8262
		40	0.5055	0.8326
		44	0	0.6410
		46	0.0225	0.8565
		48	0.0242	0.9039
Cue-once	Irregular	2	0	0.2706
		8	0	0.3548
		27	0	0.1043
		33	0.1154	0.9177
		42	0.0043	0.1652
		45	—	0.0871

4.3.2 Results and Discussion

Some of the transplanted ancestors were unable to survive the transplant to the new environment. An explanation for this phenomenon is that, in at least some of the cases that were individually analyzed, the algorithms of the organisms were so tuned to their ancestral environments that they could not survive the environmental perturbations of the new environment. These organisms are, in a sense, over-fitted to the problem posed by their native environments. One aspect of this problem is that, as observed in the genome analysis of preceding sections, some organisms tie their replication to their path-following algorithm. If these organisms are unable to navigate the path of the new environment, they will also be unable to replicate, and so will die early in evolution. The majority of the transplanted ancestors survived in all three transplant setups (8 of 10 for single-direction to right-left turn paths; 9 of 10 for right-left turn to cue-once paths; 6 of 10 for cue-once to irregular paths). All reported results and discussion that follow relate only to the populations that survived the transplant; the populations that died have been dropped and are disregarded in the results and discussion.

To have a more complete picture of how the evolved capabilities from the ancestral environment affect evolution in the transplant environment, I made a baseline measurement of the average maximum task quality (AMTQ) of the surviving transplanted ancestors in the new environments. I injected each transplant ancestor into a full population (60×60 grid, 3600 organisms), using the new environment, and allowed the population to execute for three generations with a zero mutation rate. The AMTQ was calculated from the values for each grid at the end of the 3-generation run, giving the ancestral AMTQ. The ancestral AMTQ measures how much of a "head start" the transplanted ancestors have, as opposed to starting evolution from the default self-replicating ancestor. Comparing this baseline value to the AMTQ at the end of the evolution experiments in the new environments gives an overview of how much the path-following performance improved over the course of the additional evolution. Table 4.4 summarizes the ancestral and evolved AMTQ of each surviving transplanted ancestor in the new environments. Some of the ancestors

could not survive in the zero-mutation test environment, but were able to survive in the actual experimental environment with mutations occurring. Organisms that have an ancestral AMTQ of 0 are able to replicate in the new environment, but are not able to move on the paths to garner task quality. This simple two-point comparison reveals a wide range of changes from the ancestral AMTQ to the AMTQ at the end of evolution. It is interesting to observe that the overall performance of the transplanted ancestor populations is, in aggregate, strong in all cases for the transplants from single-direction to right-left turn paths (minimum AMTQ value of 0.5773), but less impressive for the other two environments. These data, however, compare within the treatment, and so cannot inform us about the possible contributions of earlier evolution. For a better understanding of the dynamics of these experiments, we need to compare these results to those of the experiments that used the default self-replicator ancestor.

For the comparisons between the transplanted ancestor experiments and the default ancestor experiments, I again examined average maximum task quality (AMTQ) over the time course of the experimental run. If earlier evolution provided building blocks of the capabilities needed for succeeding in the new environment, the performance of the transplanted ancestors populations should be significantly better than that of the populations that evolved from the default ancestor. Figure 4.13 shows the AMTQ over time of the two treatments for the right-left turn paths. The AMTQ distributions for the transplanted ancestors and default ancestors are significantly different (Mann-Whitney U-test, $p = 6.765 \times 10^{-9}$). This difference is apparent relatively early in evolution, at 10,000 updates into the runs (Mann-Whitney U-test, $p = 2.471 \times 10^{-6}$). The transplanted ancestor populations perform demonstrably better than the default ancestor populations at the end of evolution: 86.1% (68 out of 79 populations) of the replicate populations had ending AMTQ values above the median ending AMTQ value for the default ancestor populations.

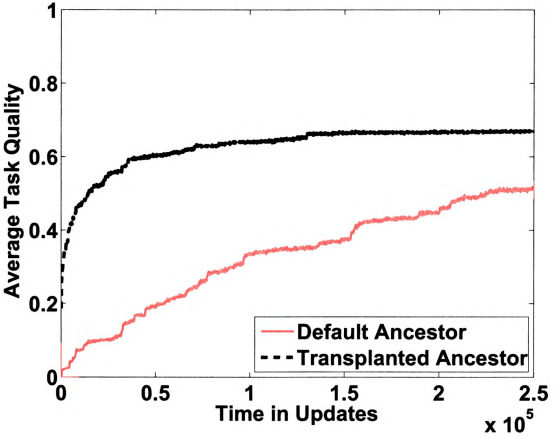


Figure 4.13: Average maximum task quality (AMTQ), evolved *vs.* default ancestor, right-left turn paths.

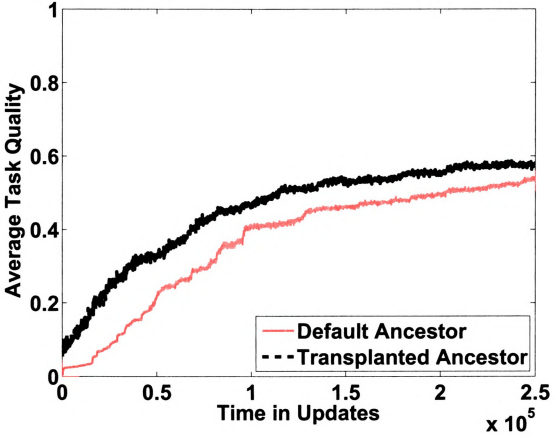


Figure 4.14: Average maximum task quality (AMTQ), evolved *vs.* default ancestor, cue-once paths.

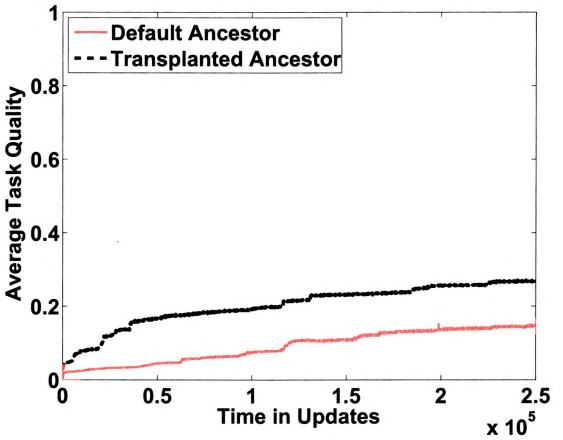


Figure 4.15: Average maximum task quality (AMTQ), evolved *vs.* default ancestor, irregular paths.

The story is similar for the cue-once paths: the transplanted ancestor populations, in aggregate, performed better than the populations with the default ancestor throughout the run (Figure 4.14). The performance difference was significant as early as 1000 updates into evolution (Mann-Whitney U-test, $p = 1.09 \times 10^{-6}$), and remained significant at the end of the run (Mann-Whitney U-test, $p = 1.56 \times 10^{-5}$). At the end of evolution, 81.8% of the populations seeded with transplanted ancestors (72 out of 88 replicate populations) had AMTQ values above the median AMTQ value of the default ancestor populations.

The populations in the irregular path experiments all had lower overall AMTQ than the populations of the other environments (Figure 4.15). The transplanted ancestors and default ancestors produced significantly different AMTQ distributions, both at 10,000 updates (Mann-Whitney U-test, $p = 9.692 \times 10^{-4}$) and at the end

of the run (Mann-Whitney U-test, $p = 6.564 \times 10^{-6}$). In addition, the populations seeded by transplanted ancestors performed significantly better than the default ancestor populations, since 83.3% of the replicate populations (50 out of 60 populations) had ending AMTQ values above the median AMTQ value of the populations seeded with the default ancestor.

The results suggest that, in the three environments of the transplant experiments, earlier evolution provided some components of the functions needed for evolving more complex traits. The populations seeded with transplanted ancestors clearly out-performed the populations seeded with the default ancestor. By comparing the capabilities needed to succeed in these different environments, we can get an intuitive sense of why the transplanted ancestors had some advantage in the new environments. Organisms that succeeded in the single-direction path environments needed to sense and respond differentially to the four states that composed the environment (nutrient, right-turn, left-turn, empty), so the organism's genome must contain instructions to handle all those states. The only salient difference between the single-direction paths and the right-left paths is that the latter contain both turn directions in the same path. Organisms transplanted from the single-direction environment had to find mechanisms to manage this difference, but they should have much of what they need already encoded in their genomes (provided they had not evolved as a "specialist" in one direction or the other in the single-direction environment). Indeed, one transplant ancestor came to the new right-left turn environment with a ready-built strategy: replicate ID 32, with an ancestral AMTQ of 0.9802, emerged from evolution in the single-direction environment with a strategy that worked immediately in the new right-left turn environment. This situation was an exception and not a rule: most organisms had far less effective solutions when initially transplanted, as shown by their ancestral AMTQ values (Table 4.4).

The shift from the cue-once environment to the irregular paths environment is the same sort of lateral step as the shift from single-direction to right-left paths. An organism that evolved to perform well in the cue-once environment is likely to have much of what it needs to do well in the irregular path environment. The irregular

path environments used all the same states as the cue-once environments, with the difference again being that the irregular path contained both turn directions instead of only one. The challenging aspect of moving from the cue-once environment to the irregular path environment was the need for "volatility" in the remembered experience, *i.e.*, the information needed to be updated at unknown intervals. This demand could be a difficult problem for an organism that evolved in the cue-once environment, where the information never needed to be changed. The results of these experiments indicate that the problem was not insurmountable, and indeed that the ability to remember one bit of information throughout its lifetime provided useful building blocks for organisms to construct shorter-term volatile memory. The detailed investigation of what those building blocks were is outside the scope of the current study, but may provide valuable insights into contingency in evolution in future work.

The cue-once environment presented a much larger change from the right-left turn environment than that between the right-left turn environment and the single-direction environment. The cue-once environment was, in fact, the first time that an organism needed to remember individual experience in order to succeed in the environment. Both the single-direction and right-left turn environments provided all the information an organism needed at each decision point: each turn was cued by a specific sense value, with no ambiguity and no reference to another event or experience. The cue-once paths, on the other hand, required an organism to somehow record its experience at the first turn, and recall that record of experience to make later decisions. This situation was further complicated by the between-grid ambiguity of the "repeat-last" state, that gave the same sensory value in all path environments. Despite these complications, the organisms that evolved to do well in a purely reflexive environment were able to transition to an environment that required memory, and quickly do better than organisms that evolved *de novo* in the environment requiring memory. These results give a general indication that even though evolving individual memory is fundamentally more challenging than evolving mechanisms that "hard code" ancestral experience into the genome, earlier

evolution of those hard-coded responses provide building blocks for more flexible use of information.

4.4 Conclusions

In the first two sets of experiments in this chapter, I investigated the evolution of a one-bit individual memory. The one bit of information that needed to be stored and reused was the binary choice of turning right or left, based on experience in the individual's life. The cue-once experiments focused on evolving life-long memory, meaning a memory that had to be stored and reused, but not changed. In these environments, a singular individual experience needed to be remembered and used for future decisions. The irregular path experiments provided the most complex experimental environment of the study, requiring frequent updating of the stored information, analogous to short-term memory.

Evolution built effective mechanisms and strategies for these differing memory demands. The cue-once environment gave rise to evolved features that resemble both reversible and permanent phenotypic plasticity, with some organisms executing different parts of their genomes depending on the current environment. The irregular path environment is clearly the most challenging of the environments presented in the current investigation, as shown by the markedly lower average maximum task quality (AMTQ) values in those experiments. In spite of the increased demands of the environment, effective solutions evolved that both capitalized on environmental regularity and functioned flexibly in both familiar and novel environments.

The story that emerges from the one-bit memory experiments gives some valuable insight into how evolution is managing the problem of memory use. In these experiments, the mechanisms for memory involve both genetic organization and volatile states in elements of the Avida organism's virtual CPU. The genomes evolve with identifiable modular features, perhaps in response to the regularities inherent in the experimental environments. Execution of those modules, however, often depends on sensory information placed in the organism's CPU (most often its registers), and

sometimes cleverly manipulated to ensure correct execution (*e.g.*, loop termination) and to permit replication. Evolution discovered the mechanisms for gathering the information from the environment, storing it for ease of future use, manipulating the information, and responding differentially to that information, according to context. Organisms leveraged these mechanisms in order to base a decision "now" on "before," meaning an individual past experience.

The results of the transplant experiments suggest that information and memory functions that evolved in simpler environments provide useful building blocks for evolving more complex memory functions. Some of the organisms that evolved in one environment evolved further to succeed in a different, more complex environment. These experiments also show that it is fundamentally more difficult to evolve a strategy for individual memory than a strategy that uses purely reflexive responses. My results also suggest that, in the absence of memory decay, it is more difficult to evolve mechanisms for short-term volatile memory than mechanisms for life-long memory. This initial foray into investigating the evolution of complex features in a new context of memory and navigation shows great promise of the potential richness of insight that may be gained in future detailed study of the emergence of complexity through evolution.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this study, I explored the evolution of fundamental memory capabilities in the context of simple navigation. The experiments demonstrated the evolution of several levels of memory, from the rudimentary memory in the "warmer-colder" gradient-following experiments presented in Chapter 2, through the "reflexes" that evolved in the simplest path-following experiments in Chapter 3, to the one-bit individual memory that evolved in the experiments in Chapter 4.

While analyzing these experiments, I found several organisms of particular interest. These organisms included the "step-counter" organism of the single-direction turn experiments, and the sample organism from the cue-once experiments that executed in a manner reminiscent of phenotypic plasticity in biological organisms.

The results of the experiments illustrate that memory and flexible behavior may evolve in even the simplest environments. Evolution capitalizes on both environmental change and environmental regularity to construct these solutions. The experiments presented here suggest, not surprisingly, that it is more difficult to evolve individual memory than to maintain "evolutionary memory" in the form of reflexes, and further that life-long individual memory evolves more readily than individual volatile, "short-term" memory. One detail of note is that the experiments of the current study did not incorporate any decay of stored information. Adding memory

decay to the experimental design would yield interesting results, and may change the balance between life-long and volatile memory. Results from the transplant experiments, where evolved organisms from one environment were transplanted into a new environment and allowed to evolve, suggest that simpler memory functions (*e.g.*, reflexes) provide building blocks for evolving more complex memory functions.

Taken as a whole, the experimental results that I present here demonstrate the evolutionary origin of simple behavioral intelligence. Organisms from these experiments were capable of gathering information from the environment, storing that information, and using the information for decisions. Moreover, organisms that succeeded in the cue-once and irregular path environments were able to use a past individual life experience to guide future decision-making.

5.2 Future Work

The experiments that I present here are only the first few steps of an exciting research initiative that has the potential for a number of important future contributions. Future directions include both extensions of the current work and new avenues of inquiry. In the following discussion, I present some of the myriad possibilities for future investigations.

5.2.1 Extensions of the Current Work

Evolving Other Components of Navigation

Natural evolution produced many impressive navigation abilities in animals. These capabilities are made up of many interwoven strategies, which are themselves made up of simpler underlying mechanisms. Memory is undoubtedly one such underlying mechanism. The experiments of the current work can easily be adjusted to focus on other component mechanisms for navigation

As has already been noted, one particularly interesting result from the single-direction path experiments was the evolution of the "step-counter" organism that

based part of its strategy on counting the number of steps it had taken along its path. This organism possesses a simple step-counting odometry mechanism. This sort of self-movement based odometry is an important aspect of many animal navigation systems. This same organism was also able to count its rotations in order to orient itself in the correct direction. Such self-referential, or idiothetic, compasses are another component of animal navigation systems. The initial results from the current study hold great promise of future insights into questions surrounding the evolution of components of navigation.

Extending these ideas one more step, we can imagine how the environments used in the current study can be adjusted so that organisms need to explore the environment to find resources (such as the "nutrients" that form the paths), and then return to their initial location as quickly or efficiently as possible. This situation sets the stage for investigating the evolution of path integration, or dead reckoning, a remarkable and nearly ubiquitous animal navigation strategy in which an animal moving through the environment maintains an incrementally updated vector back to a reference location (*e.g.*, its nest), allowing the animal to return to that reference location by the most direct route possible at any point in its journey. There is a rich collection of behavioral evidence concerning this ability in various animals, and several different models of how the mechanism may work have been presented (see, for example, Mittelstaedt (1985), Müller and Wehner (1988), Hartmann and Wehner (1995)). How evolution produced such a capability is, however, an open question. Experiments such as those I present in the current work have the potential to contribute important insights in that discussion.

Associative Memory and Learning

Associative learning can be described as the process by which animals learn about cause-and-effect relationships between events, and then behave appropriately (Rescorla, 1988; Shettleworth, 1998). Associative learning has been studied extensively, as both classical/Pavlovian conditioning and operant/instrumental conditioning, across a wide range of animal species.

The environments of the path-following experiments can be used to study the evolution of associative memory. Instead of having fixed return values for the sign-post states on the paths, we can generate a random number for the state return value each time a state grid is given to an organism. This random value would persist within the organism's lifetime, but would change for the offspring's generation. In this way, we can simulate the arbitrary stimulus that is important to associative memory and learning.

We can also vary the relationship between the cue and the target. For true associative memory, the organisms should be able to associate arbitrary features of their surroundings with their desired goal. So the cue might be prompting a turn in the paths, or it might indicate that the food source is a certain distance ahead, regardless of what else the organisms have seen in the interim. This is a far more complex form of associative memory, but it is clearly very powerful.

Investigating Plasticity

In section 3.2.2, I introduced the dynamic plasticity ratio (DPR), a measure of the execution-level plasticity of a genome. Intuitively, plasticity should aid evolvability: more flexibility at the genomic level ought to facilitate change, for example, in response to environmental change. The DPR analysis of section 3.2.2 did not supply any solid evidence about the benefits or pitfalls of such plasticity. The results of the transplant experiments in section 4.3 hint at some interesting information to be gleaned. In all three experimental environments, the populations that evolved from transplanted ancestors, that already had some capabilities, out-performed the populations that evolved from the simple self-replicator default ancestor. These results have interesting implications for looking at evolvability: in addition to having some capabilities that may have provided building blocks for more complex traits, the transplanted ancestors also had, in general, more flexible genomes than the default ancestor.

It is possible that the DPR, as implemented in the current study, failed to capture some aspects of execution-level plasticity, or the implications of that information

may not be as straightforward as I initially thought. Careful review of how the ratio is calculated and what that information may be reflecting can help improve the information these analyses provide, further illuminating topics relating to plasticity, evolvability, and intelligence.

5.2.2 Future Directions

My focus in the projects presented here was on *how* memory evolves—what traits evolve, and how they build on each other. One future direction that needs to be addressed concerns *why* those traits are favored by selection. What selective pressures encourage the evolution of memory? Such an inquiry will examine three interrelated questions: (1) Can we find adaptive steps in the evolution of memory and learning that are contingent on earlier steps? (2) How much is this capability worth to organisms, that is, how much are organisms willing to pay for the capability? (3) Which environments provide the selective pressures that evolve memory and learning? I have two over-arching, tightly coupled goals for future experiments in this direction: to gain qualitative insights into contingency and convergence in evolving memory and learning, and to derive mathematical models relating to contingency and convergence.

Another important future direction relates to evolving complexity. This topic was addressed in part by the transplant experiments in section 4.3. The issue of evolving complex features is a key topic in evolutionary biology, and contributions in that area are potentially important. The following discussion outlines ideas about how I can approach the issues of contingency, convergence, and complexity.

Selecting for the Building Blocks of Memory

Historical Contingency in Evolving Memory. Historical contingency in evolution concerns how accidental changes to the genetics of a population shape the path of future evolution; steps in evolution are thus dependent on prior history. Debate on this subject revolves around the tension between natural selection and random processes. Blount *et al.* describe this "profound tension between random and deter-

ministic processes" (Blount, Borland, & Lenski, 2008, p. 7899). Natural selection is systematic, working to adapt populations to their enviroments; random processes appear in mutations, and even beneficial mutations may disappear through random drift. Future evolutionary possibilities may be contingent on prior evolutionary history, due to the interplay between these random and deterministic processes (Blount et al., 2008).

There are varying viewpoints regarding historical contingency in evolution. Stephen Jay Gould argued that historical contingencies make evolution unpredictable, and that if we were to replay the "tape of life," and restart evolution from some point in the past, the resulting world would be dramatically different from the one that exists today (Gould, 1989). Gould saw the contributions of deterministic and random processes as they relate to general form and details:

> Invariant laws of nature impact the general forms and functions of organisms.... When we set our focus upon the level of detail that regulates most common questions about the history of life, contingency dominates and the predictability of general form recedes into an irrelevant background.... Charles Darwin recognized this central distinction between "laws in the background" and "contingency in the details". (Gould, 1989, pp. 289–290)

John Maynard Smith described evolution as "a series of historical accidents, subject to engineering constraints on the one hand, to the conservatism of development on the other" (Maynard Smith, 1986, p. 45). Simon Conway Morris took the opposing standpoint, arguing that natural selection converges on the same adaptations, despite accidents of history; in Conway Morris' view, replaying evolution may uncover many similarities in traits; he saw convergence, not contingency, as evolution's primary theme (Conway Morris, 2003).

Large scale historical contingency experiments are clearly impossible, but smaller scale experiments can achieve amazing results. Blount *et al.* (2008) recently reported their results on appearance of novel traits in an experimental population of *E. coli*. The authors found that the evolution of the studied trait (the capacity to exploit

citrate) was strongly affected by historical contingencies, and they traced the ability to three particular mutations. Blount *et al.* state

> ...our study shows that historical contingency can have a profound and lasting impact under the simplest, and thus most stringent, conditions in which initially identical populations evolve in identical environments. Even from so simple a beginning, small happenstances of history may lead populations along different evolutionary paths. A potentiated cell took the one less traveled by, and that has made all the difference. (Blount et al., 2008, p. 7905)

Experiments like this are important because they provide glimpses of the intricate interactions within evolution and the intersections where evolution took one path instead of another. However, these types of studies are difficult to conduct with living organisms. Digital evolution offers opportunities for us to replay evolution at will.

The techniques involved in tracing historical contingency in Avida revolve around taking data snapshots produced at intervals (*e.g.*, every 50,000 updates during a run of 1 million updates) in Avida experimental runs and restarting evolution at different time points. The time points are selected based on the appearance of traits or behaviors of interest. This equates to replaying the tape of the population's shared history; restarting farther back in time (*i.e.*, closer to the beginning of the run) means there is less shared history. Other aspects of historical contingency in Avida explore pathways to features. In these tests, we ask the question "can the feature evolve?" from any point along the lineage of the final dominant (the most abundant genotype in the population at the end of an experimental run).

I anticipate addressing three main questions with the historical contingency experiments.

1. What are the building blocks of memory and learning? To look at this question, I can select multiple time points for restarting evolution, before and after an interesting behavioral change. Looking at what is happening on both sides of

the change can help tease apart what makes the new behavior beneficial, and may provide insight into the evolutionary steps needed to build the trait. We can also zero in on individual mutations to find out if the probability of evolving a trait improves gradually, or jumps suddenly due to a single evolutionary event.

2. What is the likelihood that memory and learning will evolve if we go back to different time points during evolution? Different replicates restarted from the same point may not evolve the same traits by the end of the run. Examining these results can provide some insight into the importance of the shared history of evolution and the chain of contingent events within that history. Results of these experiments may also reveal convergent solutions in evolving memory and learning, and provide data to support mathematical modeling of contingency and convergence in the shared evolutionary history of the experiments.

3. How does the behavior at the end of a run change depending on how far back in history we restart evolution? This experiment looks directly at the issue of contingency: how do different events during evolution affect the outcome? Another interesting variation on this experiment can provide information about population-level influences as opposed to individual-level effects. Restarted runs can be seeded with either the full population from that time point, or with just the organism in the final dominant's lineage. Using both these setups, I can see if there are population effects that help bring about the end behavior, or if the important events are mostly in the dominant lineage itself.

Costs of Memory. Mery and Kawecki's experiments (Mery & Kawecki, 2002, 2003, 2004, 2005) highlight the fascinating issue of the evolutionary costs of learning. I am interested in exploring the balance point between costs and benefits of memory and learning: how much will organisms pay for a useful trait?

To probe the question of how much organisms are willing to pay for useful memory, we can provide an instruction that does exactly what the Avidians need it to do. If the cost of that instruction is no higher than the cost of other instructions,

the Avidians get the capability essentially for free. Then, we gradually increase the cost of that instruction (*e.g.*, by making the instruction take a long time to execute). At some point, the cost of the special instruction will become too high.

For example, recall the experiments from the preceding chapters. Organisms in those environments must evolve to remember what direction they last turned. Instead, we can make life easier for them. We can provide an instruction that always turns the organism to face the correct way along the path; call the instruction *rotate-correct*. This instruction keeps the Avidian on the correct path at all times, and so would be advantageous. Without *rotate-correct*, organisms have to discover their own mechanisms for remembering their last action. However, is there a price that is too high, despite *rotate-correct*'s utility? I expect that there is such a balance point. Evolution can still assemble a sequence of instructions that have most of the same functionality as the special *rotate-correct* instruction. When the cost of the special instruction reaches some level, it will be less costly for evolution to search for a comparable mechanism than to continue to pay the high cost of *rotate-correct*. If this is the case, we will be able to accurately predict the cost that organisms are willing to pay for memory, which in turn provides a measure of what those capacities are worth.

Influence of Environments in Evolving Memory. The role played by environment in evolving memory is an important theme in the work I present here. The influence of the environment is certainly a key issue in the evolution of intelligence in general, and the evolution of memory and learning in particular.

One way that we can determine which environments encourage the evolution of memory is to start with an organism that has the ability to remember key information. We can seed populations in different environments with this organism, and adjust the cost of the instruction that provides the memory capability. We can then see under which environmental conditions the ability persists at given costs, by observing if the memory capability persists over time under the pressure of mutations. If the memory capability is lost, then the environment in question lacks the selective

pressures to support memory. Similar to the preceding experiment, this experiment addresses costs and benefits; this experiment, however, focuses on the part that environment plays in the cost/benefit equation. Memory is a complex capability, and so must have costs associated with it. For some environments—perhaps more complex ones—the benefits of possessing and using the complex capability will outweigh the expense of maintaining that complex capability. In other environments, however, the complex ability may not be as advantageous, and so will disappear over time. This process will lead to some mathematical models of trade-offs between costs and benefits. Ideally, we will be able to make predictions of how likely an environment is to give rise to memory.

Exploring the Evolution of Complex Features

Avida is an invaluable tool for experiments in the evolution of complex features: all details of the evolutionary record are available for analysis, without any "missing links" in the genealogy from initial ancestor to descendants (Lenski et al., 2003) The transplant experiments of section 4.3 established a foundation for using these new simple navigation environments for experiments in evolving complexity.

To date, most Avida experiments have been done using the nine logic task environment (Logic-9) (Ofria et al., 2002). In this environment, organisms are rewarded for evolving to execute one- and two-input logic operations on bit strings. Lenski *et al.* (2003) examined the evolutionary origin of the equals (EQU) operation, the most complex logic operation in the Logic-9 environment. However, it is unclear how well the results from the Logic-9 environment generalize. The new environments in the current study supplement the existing Avida environments. Detailed experiments such as those found in Lenski *et al.* (2003) could be repeated in these new navigation environments, to shed more light on how evolution builds complex features.

The work I have presented clearly demonstrates that investigating simple behavioral intelligence can provide real insights. These insights relate both to the evolution of those simple capacities, and how those simpler functions may coalesce into more complex faculties. This foundational research is of interest not only to

those considering the evolution of natural intelligence, but also for the promise it holds for improving future computational systems. The ultimate goal of my research is two-fold: to add to the body of knowledge of evolution, and to apply that knowledge to build artificial systems with the same robustness and flexibility evident in so many of the creatures of the natural world.

APPENDIX

Appendix A

Annotated Avida Code of Selected Evolved Genomes

A.1 Avida Instruction Set for Experiments

Table A.1 lists the full set of Avida instructions used for all experiments in Chapters 3 and 4 of this study. The experimental instruction set used the 26-instruction default instruction set, an additional existing comparison instruction (*if-grt-0*), and 6 new instructions that we added for these experiments.

For additional information and detail regarding the operation of the instructions or the Avida virtual hardware, see Ofria and Wilke (2004).

Notes on the Instruction Descriptions:

- The notation ?XY?, where XY is a register (AX, BX, or CX) or head (IP, FH, RH, WR) signifies that the CPU component specified is the default for the instruction. The component may be modified by a no-op (NOP) label immediately following an instruction.

- Complement: The complement of a NOP label is the next label, in alphabetical order, looping around at the end of the list: nop-A -> nop-B -> nop-C -> nop-A.

- Modification by NOP labels: NOP labels can alter the operation of instructions by the component of the virtual CPU the instruction operates on. This change occurs when the NOP label immediately follows certain instructions,

as described below. When referring to a register, nop-A indicates AX, nop-B is BX, and nop-C is CX. When referring to a head, nop-A indicates the instruction pointer (IP), nop-B is the read-head (RH), nop-C is the write-head (WH). The flow-head (FH) can be used only by default.

Table A.1: Avida Instruction Set Used in Current Experiments.

Instruction	**Description**
Default Instruction Set	
nop-A	Does nothing by itself, but will modify the operation of an instruction by changing the CPU component the instruction operates on (see notes, above).
nop-B	See description of nop-A, above.
nop-C	See description of nop-A, above.
if-n-equ	Compares the contents of ?BX? to the contents of its complement. If the two values are equal, the next instruction (after a modifying NOP label) is executed. Otherwise, the next instruction is skipped.
if-less	Compares the contents of ?BX? to the contents of its complement. If the value in ?BX? is less than the value in the complement, the next instruction (after a modifying NOP label) is executed. Otherwise, the next instruction is skipped.
pop	Remove the top item from the active stack and place it in ?BX?.
push	Copy the current contents of ?BX? and place it as a new entry at the top of the active stack.
swap-stk	Change the active stack.
swap	Exchange the contents of ?BX? with its complement.
shift-r	Reads the contents of ?BX? and shifts all bits one position to the right. This divides the previous value by 2, rounding down.
	Continued on next page

Table A.1 – continued from previous page

Instruction	Description
shift-l	Reads the contents of ?BX? and shifts all bits one position to the left. Bits in excess of 32 are truncated. For values needing less than 32 bits for their representation, this multiplies the prior value by 2.
inc	Increment the value in ?BX? by 1.
dec	Decrement the value in ?BX? by 1.
add	Add the contents of BX and CX and place the result in ?BX?.
sub	Subtract the contents of BX and CX and place the result in ?BX?.
nand	Perform a bitwise NAND of the contents of BX and CX and place the result in ?BX?.
IO	Output the value of ?BX?, check for any tasks performed, and input new value into ?BX?.
h-alloc	Allocate new memory for the organism, up to the maximum it is allowed to use for its offspring.
h-divide	Divides off an offspring. The parent organism keeps the state of its memory to the read-head. The offspring's memory is initialized to the contents of memory between the read-head and the write-head. Any memory past the write-head is removed.
h-copy	Copies the contents of the organism's memory at the position of the read-head to the position of the write-head. If the copy mutation rate is non-zero, a random instruction will be placed at the write-head according to the mutation probability.

Continued on next page

Table A.1 – continued from previous page

Instruction	Description
h-search	Reads the label that follows the instruction and finds the location of a complement label in the code. BX is set to the distance from the current IP position to the complement, and CX is set to the size of the label. The flow-head is placed at the beginning of the complement label. If no label follows, set BX and CX to zero, and place the flow-head on the instruction immediately following the *h-search*.
mov-head	Jumps the ?IP? to the position of the flow-head.
jmp-head	Reads the value in CX and moves the ?IP? that fixed amount in the organism's memory.
get-head	Copy the current position of the ?IP? to CX.
if-label	Reads in the label following the instruction. If the label's complement was the most recently copied series of instructions, execute the next instruction, otherwise skip the next instruction.
set-flow	Move the flow-head to the position in memory given by the value in CX.
if-grt-0	Compares the contents of ?BX? to 0. If the value in ?BX? is less than 0, the next instruction (after a modifying NOP label) is executed. Otherwise, skip the next instruction.
New Instructions for Current Experiments	
sg-move	Move to the cell the organism is currently facing.
sg-rotate-l	Rotate one 45° turn left (counter-clockwise).
sg-rotate-r	Rotate one 45° turn right (clockwise).
sg-sense	Returns the value of the state in the current cell.

Continued on next page

Table A.1 – continued from previous page

Instruction	Description
if-grt-X	Compares the contents of BX to a fixed value determined by the modifying NOP label (default = 1, nop-A = -1, nop-B = 2, nop-C = 4). If BX is greater than the value, execute the next instruction (after the NOP label), otherwise skip the next instruction.
if-equ-X	Compares the contents of BX to a fixed value, as in *if-grt-X*, above. If the value in BX is equal to the value, execute the next instruction (after the NOP label), otherwise skip the next instruction.

A.2 Annotated Avida Code of Evolved Organisms

The following sections contain the complete genomes of the example organisms discussed in Chapters 3 and 4. For each example organism, I provide the full genome, highlighting the instructions that I will explain in more detail. The detailed explanations focus on the operation of the code for the instructions that are important in the path-following task. I also include some additional information on the genomes, including the organism's gestation time (number of instructions needed to replicate) and genome sizes.

A.2.1 Single-direction Paths Example Organism

The organism below evolved in the single-direction turn environment. The organism has two main movement modules, one that is primarily concerned with moving through right turns ("Module A"), and another that is most important for left turns ("Module B"). Both of these code sections are executed, whether the organism is

moving on a right-turn path or a left-turn path; the details of the execution flow ensure correct execution of all the code for both path directions.

The organism uses Module A, the section that is primarily concerned with right turns, in an interesting way. When on a right-turn path, the organism uses this module to count its steps to the end of the path; the loop exit occurs when the organism reaches the end of the path. When on a left turn path, the organism skips executing the *sg-move* instruction, and counts one-eighth-turns instead of steps. Execution exits the loop when the organism has completed four one-eighth-turns, reversing its facing. The code segment that follows this module again reverses the organism's facing with four consecutive 45-degree turns to the right.

Module B, concerned primarily with left turns, contains a nested copy loop. Execution enters the copy loop only when the organism steps off the path into an empty cell. On the paths experienced during evolution, the copy loop is executed exclusively when the organism is executing on a left-turn path. When on a right-turn path, the organism's replication code is more distributed throughout the genome, with a number of *h-copy* instructions sprinkled in various places in the code. In all cases, the same *h-divide* instruction triggers dividing off the completed offspring.

Organism Information:

- Gestation Time: 1779
- Genome Size: 185
- Copied Size: 185
- Executed Size: 156

Table A.2 shows the complete genome of the organism. Instructions that are shown in ***bold, italicized, red text*** are important for path-following. I discuss the operation of the path-following instructions in more detail in Table A.3.

Table A.2: Genome, Example Organism, Single-direction Paths Experiments.

Position	Instruction	Position	Instruction	Position	Instruction
0	if-grt-0	41	jmp-head	82	inc
1	if-grt-0	42	h-divide	83	dec
2	IO	43	nop-B	84	add
3	if-less	44	get-head	85	if-grt-0
4	if-n-equ	45	get-head	86	h-search
5	if-equ-X	46	sg-sense	87	push
6	nop-A	47	if-grt-0	88	shift-r
7	jmp-head	48	if-less	89	sub
8	h-search	49	shift-r	90	if-less
9	push	50	inc	91	nand
10	get-head	51	get-head	92	pop
11	sub	52	h-alloc	93	sub
12	if-grt-0	53	swap-stk	94	swap-stk
13	h-search	54	IO	95	swap
14	IO	55	if-label	96	sg-sense
15	get-head	56	h-divide	97	if-equ-X
16	set-flow	57	if-grt-X	98	pop
17	h-search	58	if-grt-X	99	swap
18	get-head	59	shift-l	100	shift-l
19	if-equ-X	60	h-divide	101	h-divide
20	set-flow	61	nop-C	102	set-flow
21	nop-A	62	h-divide	103	nop-B
22	get-head	63	nop-C	104	nop-B
23	IO	64	sg-sense	105	push
24	swap-stk	65	if-label	106	jmp-head
25	add	66	if-label	107	sg-move
26	if-grt-0	67	set-flow	108	if-equ-X
27	if-n-equ	68	nop-A	109	if-grt-X
28	nand	69	if-grt-0	110	jmp-head
29	if-grt-0	70	push	111	swap
30	if-less	71	swap-stk	112	add
31	if-equ-X	72	if-equ-X	113	swap-stk
32	mov-head	73	sg-sense	114	if-n-equ
33	nop-A	74	sub	115	shift-l
34	push	75	nop-B	116	sub
35	nand	76	nop-B	117	***h-search***
36	if-label	77	IO	118	***sg-rotate-r***
37	h-search	78	h-alloc	119	***if-grt-0***
38	nop-A	79	sg-move	120	***nop-C***
39	swap-stk	80	mov-head	121	***h-copy***
40	swap	81	nop-C	122	***h-copy***

Continued on next page

Table A.2 – continued from previous page

Position	Instruction	Position	Instruction	Position	Instruction
123	***sg-sense***	144	if-less	165	swap-stk
124	***nop-C***	145	shift-l	166	***h-search***
125	***jmp-head***	146	nand	167	***sg-move***
126	***sg-rotate-l***	147	if-equ-X	168	***sg-sense***
127	***if-equ-X***	148	shift-l	169	***nop-B***
128	***get-head***	149	***sg-rotate-r***	170	***if-n-equ***
129	***sg-move***	150	push	171	***sg-rotate-l***
130	***inc***	151	nop-B	172	***if-equ-X***
131	***if-n-equ***	152	add	173	***pop***
132	***mov-head***	153	***sg-rotate-r***	174	***if-less***
133	sg-move	154	inc	175	***h-search***
134	IO	155	if-equ-X	176	***if-label***
135	sg-move	156	h-copy	177	***nop-C***
136	if-grt-0	157	if-label	178	***h-divide***
137	***sg-rotate-r***	158	nop-B	179	***h-copy***
138	h-alloc	159	if-equ-X	180	***mov-head***
139	if-grt-0	160	add	181	set-flow
140	swap-stk	161	set-flow	182	push
141	nand	162	get-head	183	swap
142	***sg-rotate-r***	163	nop-B	184	nop-A
143	IO	164	set-flow		

Table A.3: Genome Detail for Example Organism, Single-direction Paths Experiments.

Position	Instruction	Comments
117	h-search	Start of Module A.
118	sg-rotate-r	Turn right 45°.
119	if-grt-0	CX > 0? True when on right-turn path.
120	nop-C	
121	h-copy	Do an extra copy when on a right-turn path.
122	h-copy	Copy (always executes).
123	sg-sense	CX=sense

Continued on next page

Table A.3 – continued from previous page

Position	Instruction	Comments
124	nop-C	
125	jmp-head	If sense was nutrient, CX=0; if sense was right, CX=2; if sense was left, CX=4
126	sg-rotate-l	Executes only when sense=nutrient; undoes right turn at top of loop.
127	if-equ-X	BX=1? True on right-turn path.
128	get-head	CX=128 (right-turn path only).
129	sg-move	Take a step. Does not execute on left-turn path.
130	inc	BX= BX+1.
131	if-n-equ	BX!=CX? Tests for loop exit conditions.
132	mov-head	Exit on right-turn path after taking 127 steps, then incrementing BX to 128. Exit on left-turn path after turning 180° without taking any steps.
137	sg-rotate-r	Turn right 45°. 1st of 4 consecutive right turns.
142	sg-rotate-r	Turn right 45°. 2nd of 4 consecutive right turns.
149	sg-rotate-r	Turn right 45°. 3rd of 4 consecutive right turns.
153	sg-rotate-r	Turn right 45°. 4th of 4 consecutive right turns. Organism's facing is now reversed; facing "backwards" on right-turn path, facing correctly on left-turn path.
	h-search	Start Module B, including a nested copy loop.
	sg-move	Take a step.
	sg-sense	BX=sense.
	nop-B	

Continued on next page

Table A.3 – continued from previous page

Position	Instruction	Comments
	if-n-equ	BX!=CX? True if sense=left, False if sense=0. Always False on right-turn path (no more turns remaining; sense=0)
	sg-rotate-l	Turn left 45°.
	if-equ-X pop	BX=1? Always False.
	if-less	BX<CX? True when sense=empty.
	h-search	Start of copy loop. Entered only when organism moves into empty cell.
	if-label nop-C	Was last copied nop-A?
	h-divide	If last copied was nop-A, divide.
	h-copy mov-head	Copy an instruction. If last sense!=empty, target is top of left-turn loop. If last sense=empty, target is top of copy loop.

A.2.2 Right-left Turn Paths Example Organism

The following organism evolved in the right-left turn environment. Although it has the longest genome of the organisms discussed here, the organism does most of the "work" with a sequence of ten instructions, used for path-following, and a short, separate copy loop. The turns in the path-following code are guarded with conditionals that ensure executing the correct action at the correct time.

Organism Information:

- Gestation Time: 1438
- Genome Size: 218

- Copied Size: 218
- Executed Size: 201

Table A.4 shows the complete genome of the organism. Instructions that are shown in ***bold, italicized, red text*** are important for path-following. I discuss the operation of the path-following instructions in more detail in Table A.5.

Table A.4: Complete Genome of Example Organism, Right-left Turn Paths Experiments.

Position	Instruction	Position	Instruction	Position	Instruction
0	nop-A	30	swap-stk	60	nop-B
1	nop-A	31	sg-sense	61	sg-sense
2	nop-A	32	inc	62	h-alloc
3	if-n-equ	33	h-divide	63	push
4	swap-stk	34	get-head	64	if-less
5	inc	35	inc	65	h-search
6	if-equ-X	36	set-flow	66	IO
7	dec	37	IO	67	nop-B
8	push	38	nop-B	68	if-equ-X
9	inc	39	sg-move	69	shift-r
10	sub	40	if-n-equ	70	if-grt-X
11	shift-l	41	shift-l	71	h-search
12	h-alloc	42	if-equ-X	72	add
13	if-n-equ	43	if-grt-0	73	shift-l
14	if-label	44	h-divide	74	h-search
15	nop-A	45	nop-C	75	add
16	if-n-equ	46	h-alloc	76	sg-sense
17	shift-l	47	pop	77	if-grt-0
18	h-copy	48	h-copy	78	h-divide
19	h-divide	49	inc	79	nop-C
20	pop	50	shift-l	80	sg-rotate-l
21	h-copy	51	swap-stk	81	dec
22	add	52	sg-rotate-l	82	if-n-equ
23	push	53	if-n-equ	83	if-less
24	add	54	sg-sense	84	if-grt-0
25	nop-B	55	h-divide	85	push
26	swap	56	nop-B	86	set-flow
27	IO	57	sg-sense	87	if-n-equ
28	swap	58	push	88	shift-l
29	nand	59	add	89	add

Continued on next page

Table A.4 – continued from previous page

Position	Instruction	Position	Instruction	Position	Instruction
90	shift-r	130	if-equ-X	169	inc
91	sg-rotate-l	131	nop-B	170	h-copy
92	shift-l	132	h-alloc	171	if-grt-0
93	nop-B	133	IO	172	IO
94	shift-r	134	sub	173	jmp-head
95	h-search	135	sub	174	if-equ-X
96	dec	136	if-grt-X	175	sub
97	swap	137	push	176	nand
98	sub	138	if-n-equ	177	push
99	h-alloc	139	sg-rotate-r	178	h-alloc
100	h-divide	140	swap-stk	179	get-head
101	IO	141	h-divide	180	IO
102	IO	142	if-equ-X	181	h-alloc
103	h-divide	143	push	182	nop-C
104	h-divide	144	swap-stk	183	swap
105	shift-r	145	push	184	if-equ-X
106	sg-rotate-l	146	sub	185	h-divide
107	set-flow	147	nand	186	sg-rotate-r
108	nand	148	h-copy	187	sg-rotate-r
109	swap	149	pop	188	*h-search*
110	nop-C	150	h-divide	189	*if-grt-X*
111	nop-A	151	get-head	190	*sg-rotate-l*
112	IO	152	nop-C	191	*h-copy*
113	push	153	h-alloc	192	*sg-move*
114	pop	154	add	193	*sg-sense*
115	sg-move	155	if-grt-0	194	*shift-r*
116	jmp-head	156	dec	195	*if-equ-X*
117	nop-C	157	sg-sense	196	*sg-rotate-r*
118	sg-rotate-r	158	nand	197	*if-less*
119	push	159	dec	198	*if-grt-0*
120	if-n-equ	160	h-copy	199	*mov-head*
121	sub	161	h-search	200	if-n-equ
122	shift-r	162	h-copy	201	shift-r
123	push	163	if-label	202	inc
124	nop-C	164	if-equ-X	203	IO
125	shift-l	165	sg-rotate-r	204	set-flow
126	h-divide	166	if-n-equ	205	jmp-head
127	nand	167	sg-rotate-r	206	h-search
128	get-head	168	if-label	207	h-copy
129	shift-l			208	if-label

Continued on next page

Table A.4 – continued from previous page

Position	Instruction	Position	Instruction	Position	Instruction
209	nop-C	212	h-divide	215	mov-head
210	if-n-equ	213	pop	216	set-flow
211	mov-head	214	get-head	217	nop-A

Table A.5: Genome Detail for Example Organism, Right-left Turn Paths Experiments.

Position	Instruction	Comments
188	h-search	Start movement loop.
189	if-grt-X	BX>1? True when sense=left
190	sg-rotate-l	Turn left 45°.
191	h-copy	Copy an instruction.
192	sg-move	Take a step.
193	sg-sense	BX=sense.
194	shift-r	BX=BX/2 (unless BX<0; then BX value is unchanged).
195	if-equ-X	BX=1? True if last sense=right, False otherwise.
196	sg-rotate-r	Turn right 45°. Executes only when last sense=right.
197	if-less	BX<CX? True when last sense=empty. False for all others states, so jump to top of loop.
198	if-grt-0	BX>0? Executes only when last sense=empty. Jumps over *mov-head* to exit loop.
199	mov-head	Exit loop when last sense=empty, otherwise jump to top of loop.

A.2.3 Cue-once Paths Example Organism

The following organism evolved in the cue-once environment. This organism has a right-turn loop, that executes in both right- and left-turn environments, and a left-turn loop that executes only in left-turn environments. The organism's replication scheme differs somewhat between right- and left-turn paths. For a right-turn path, nearly all copying takes place in the right-turn loop itself, and there is no label-checking to make sure the whole genome is copied. For left-turn paths, the organism does some copying in the right-turn loop, and finishes the process in a copy loop nested in the left-turn loop. This copy loop has the usual label-checking strategy with *if-label*. The replication strategy for the right-turn paths (*i.e.*, the strategy that does not use label-checking) seems risky; however, the strategy works because that loop terminates only when a left turn or an empty cell is sensed.

Organism Information:

- Gestation Time: 1168
- Genome Size: 136
- Copied Size: 136
- Executed Size: 73

Table A.6 shows the complete genome of the organism. Instructions that are shown in ***bold, italicized, red text*** are important for path-following. I discuss the operation of the path-following instructions in more detail in Table A.7.

Table A.6: Complete Genome of Example Organism, Cue-once Paths Experiments.

Position	Instruction	Position	Instruction	Position	Instruction
0	nop-A	5	h-divide	10	if-less
1	nop-A	6	if-equ-X	11	shift-r
2	h-alloc	7	get-head	12	set-flow
3	if-grt-X	8	set-flow	13	dec
4	set-flow	9	nop-A	14	inc

Continued on next page

Table A.6 – continued from previous page

Position	Instruction	Position	Instruction	Position	Instruction
15	sg-sense	56	*sg-rotate-r*	96	*sg-rotate-r*
16	swap	57	*sg-move*	97	*sg-move*
17	get-head	58	*h-copy*	98	*sg-sense*
18	set-flow	59	*sg-sense*	99	*swap-stk*
19	nop-A	60	*if-grt-X*	100	*if-less*
20	nop-A	61	*h-copy*	101	*h-search*
21	if-label	62	*if-grt-X*	102	*h-copy*
22	shift-l	63	*dec*	103	*if-label*
23	pop	64	*if-n-equ*	104	*nop-C*
24	mov-head	65	*if-equ-X*	105	*h-divide*
25	nop-C	66	*mov-head*	106	*mov-head*
26	sg-sense	67	sg-rotate-l	107	if-grt-X
27	sub	68	h-search	108	nop-C
28	sub	69	push	109	sg-rotate-r
29	if-grt-0	70	h-copy	110	sg-sense
30	inc	71	swap	111	dec
31	h-divide	72	nand	112	if-label
32	add	73	h-copy	113	IO
33	if-less	74	get-head	114	IO
34	swap	75	h-search	115	if-grt-X
35	get-head	76	*h-divide*	116	set-flow
36	sg-sense	77	h-alloc	117	IO
37	nop-C	78	h-alloc	118	if-grt-X
38	push	79	shift-l	119	set-flow
39	add	80	h-divide	120	shift-l
40	if-n-equ	81	if-equ-X	121	if-grt-0
41	nop-A	82	jmp-head	122	if-equ-X
42	IO	83	inc	123	IO
43	inc	84	nop-C	124	if-grt-X
44	h-alloc	85	shift-l	125	if-less
45	h-alloc	86	nop-B	126	set-flow
46	swap-stk	87	swap-stk	127	IO
47	if-grt-X	88	get-head	128	if-grt-X
48	sg-sense	89	h-alloc	129	set-flow
49	h-alloc	90	*h-search*	130	if-equ-X
50	h-copy	91	*sg-rotate-l*	131	if-n-equ
51	add	92	*if-equ-X*	132	nop-C
52	set-flow	93	*sg-move*	133	get-head
53	shift-l	94	*if-grt-0*	134	dec
54	*h-search*	95	*if-less*	135	nop-A
55	*if-n-equ*				

Table A.7: Genome Detail for Example Organism, Cue-once Paths Experiments.

Position	Instruction	Comments
54	h-search	Begin Module A.
55	if-n-equ	BX!=CX? True when last sense=right or repeat last.
56	sg-rotate-r	Turn right 45°. Executes when last sense=right or repeat last, skipped otherwise.
57	sg-move	Take a step.
58	h-copy	Copy an instruction.
59	sg-sense	BX=sense.
60	if-grt-X	BX>1? True when last sense=right or left, False otherwise.
61	h-copy	Copy an instruction.
62	if-grt-X	BX>1? True when last sense=right or left, False otherwise.
63	dec	BX=BX-1.
64	if-n-equ	BX!=CX? True for all states except nutrient.
65	if-equ-X	BX=1? True when last sense=right or repeat last.
66	mov-head	Skip and exit loop if last sense=left or empty, otherwise jump to top of loop.
76	h-divide	Divide here when on right-turn paths.
90	h-search	Start left turn and nested copy loop.
91	sg-rotate-l	Turn left 45°.
92	if-equ-X	BX=1? True when last sense=repeat last, False otherwise.
93	sg-move	Take a step.
94	if-grt-0	BX>0? True if last sense=left turn or repeat last, False otherwise.

Continued on next page

Table A.7 – continued from previous page

Position	Instruction	Comments
95	if-less	BX<CX? True if last sense=nutrient, False if last sense=repeat last.
96	sg-rotate-r	Turn right 45°. Executes when last sense=nutrient (undoes turn at top of loop).
97	sg-move	Take a step.
98	sg-sense	BX=sense.
99	swap-stk	Change active stack. No effect on execution outcome.
100	if-less	True if last sense=empty, False otherwise.
101	h-search	Begin copy loop. Enter only if last sense=empty.
102	h-copy	Copy an instruction.
103	if-label	Was last copied nop-A? If so, execute instruction following NOP.
104	nop-C	
105	h-divide	Divide.
106	mov-head	Remain in loop while sense!=empty or not done copying.

A.2.4 Irregular Paths Example Organism

This organism evolved in the irregular paths environment. It operates by differentially setting jump targets (*i.e.*, the value of the flow-head), depending on the current state value given by using the *sg-sense* instruction. The organism has distinct sections of code for right- and left-turn path segments, but can easily change execution from one direction to the other.

Organism Information:

- Gestation Time: 2660
- Genome Size: 179

- Copied Size: 179
- Executed Size: 155

Table A.8 shows the complete genome of the organism. Instructions that are shown in ***bold, italicized, red text*** are important for path-following. I discuss the operation of the path-following instructions in more detail in Table A.9.

Table A.8: Complete Genome of Example Organism, Irregular Paths Experiments.

Position	Instruction	Position	Instruction	Position	Instruction
0	***nop-A***	33	if-less	66	get-head
1	h-alloc	34	h-search	67	nop-C
2	nop-B	35	jmp-head	68	if-less
3	mov-head	36	if-equ-X	69	h-search
4	nop-B	37	h-copy	70	pop
5	add	38	shift-l	71	set-flow
6	set-flow	39	h-search	72	dec
7	swap-stk	40	IO	73	nand
8	inc	41	if-grt-0	74	if-n-equ
9	add	42	sg-sense	75	h-divide
10	nop-A	43	if-equ-X	76	if-less
11	nop-A	44	sg-sense	77	get-head
12	set-flow	45	nand	78	h-search
13	h-copy	46	if-grt-X	79	nop-A
14	if-less	47	if-less	80	nop-C
15	get-head	48	get-head	81	mov-head
16	jmp-head	49	push	82	nop-C
17	IO	50	inc	83	swap
18	if-less	51	IO	84	IO
19	add	52	h-search	85	if-less
20	set-flow	53	push	86	pop
21	add	54	inc	87	if-grt-X
22	swap-stk	55	IO	88	h-search
23	dec	56	inc	89	get-head
24	h-search	57	if-label	90	dec
25	swap	58	if-grt-0	91	push
26	h-divide	59	add	92	***h-search***
27	sg-sense	60	add	93	***shift-l***
28	if-n-equ	61	h-alloc	94	***sg-move***
29	if-n-equ	62	nop-C	95	***sg-sense***
30	nop-A	63	add	96	***sub***
31	sub	64	shift-l	97	***if-less***
32	h-alloc	65	if-grt-0	98	***swap***

Continued on next page

Table A.8 – continued from previous page

Position	Instruction	Position	Instruction	Position	Instruction
99	*h-divide*	126	if-n-equ	153	if-grt-0
100	*dec*	127	shift-r	154	nop-C
101	*if-less*	128	sg-rotate-r	155	shift-l
102	*mov-head*	129	IO	156	nand
103	*push*	130	nop-C	157	swap-stk
104	*if-label*	131	*h-search*	158	h-copy
105	*nop-C*	132	*sg-rotate-r*	159	if-n-equ
106	*shift-r*	133	*if-label*	160	swap
107	*nop-A*	134	*nop-B*	161	if-grt-0
108	*sg-rotate-l*	135	*add*	162	nop-B
109	*if-equ-X*	136	*nop-B*	163	if-label
110	*if-less*	137	*if-less*	164	push
111	*mov-head*	138	*sg-sense*	165	h-search
112	nand	139	*sub*	166	h-search
113	shift-r	140	*sg-move*	167	h-search
114	swap	141	*nand*	168	h-copy
115	if-grt-X	142	*sg-sense*	169	if-label
116	swap	143	*if-equ-X*	170	nop-C
117	shift-r	144	*mov-head*	171	nop-A
118	if-n-equ	145	*if-grt-0*	172	h-divide
119	nop-B	146	*set-flow*	173	mov-head
120	shift-l	147	*inc*	174	jmp-head
121	swap-stk	148	*jmp-head*	175	swap
122	if-equ-X	149	*push*	176	pop
123	swap	150	*sg-rotate-l*	177	nop-A
124	if-equ-X	151	*if-n-equ*	178	nop-B
125	nop-A	152	*mov-head*		

Table A.9: Genome Detail for Example Organism, Irregular Paths Experiments.

Position	Instruction	Comments
0	nop-A	Jump to here from Module B (position 152) if last sense=left turn.
92	h-search	Start of Module A.
93	shift-l	BX=BX×2 (no important effect on execution).
94	sg-move	Take a step.
95	sg-sense	Put the current cell state in BX.

Continued on next page

Table A.9 – continued from previous page

Position	Instruction	Comments
96	sub	BX=BX - CX; CX usually 0.
97	if-less	BX < CX? Usually not.
98	swap	Does not execute.
99	h-divide	No effect here.
100	dec	BX=BX-1.
101	if-less	True if sense=nutrient.
102	mov-head	Nested straight-ahead loop. Stay in nested loop as long as moving straight on path; otherwise, skip this instruction.
103	push	Save BX to stack, but never use it later.
104	if-label	Was last copied nop-A? If so, execute instruction following NOP, otherwise skip.
105	nop-C	
106	shift-r	AX=AX/2; no effect on execution.
107	nop-A	
108	sg-rotate-l	Turn left 45°.
109	if-equ-X	BX=1? True if last sense= =right turn.
110	if-less	Execute when last sense=right turn; evaluates to False, so skip next instruction and exit loop.
111	mov-head	Stay in loop if last sense=left turn or repeat last.
131	h-search	Start of Module B.
132	sg-rotate-r	Turn right 45°.
133	if-label	Nothing important for path-following until the move.
134	nop-B	
135	add	
136	nop-B	
137	if-less	BX<CX?

Continued on next page

Table A.9 – continued from previous page

Position	Instruction	Comments
138	sg-sense	BX=sense.
139	sub	
140	sg-move	Take a step.
141	nand	BX=BX NAND CX, but no impact on movement.
142	sg-sense	BX=sense.
143	if-equ-X	BX=1? True if sense=repeat last; return to top of this loop.
144	mov-head	Stay in this loop if sense=repeat last; otherwise, skip this instruction
145	if-grt-0	BX>CX? True when sense=left turn.
146	set-flow	Set flow-head to 0 (only when sense=left turn).
147	inc	BX=BX+1.
148	jmp-head	No effect (CX=0).
149	push	
150	sg-rotate-l	Turn left 45°.
151	if-n-equ	True for all states except empty (sense=-1).
152	mov-head	Execute for all states but empty, but targets differ. Stay in this module if last sense=repeat last or right turn); return to beginning of genome if sense=left turn; skip this instruction if sense=empty, and exit loop.

BIBLIOGRAPHY

BIBLIOGRAPHY

Adami, C., Ofria, C. A., & Collier, T. C. (2000). Evolution of biological complexity. *Proceedings of the National Academy of Science, 97*, 4463–4468.

Adler, J. (1966, August 12). Chemotaxis in bacteria. *Science, 153*(3737), 708-716.

Adler, J. (1975). Primitive sensory and communication systems: the taxes and tropisms of micro-organisms and cells. In M. J. Carlile (Ed.), (p. 91-100). London: Academic Press.

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist, 30*, 441–451.

Beckmann, B., McKinley, P. K., & Ofria, C. (2007). Evolution of an adaptive sleep response in digital organisms. In *Advances in Artificial Life - Proceedings of the 9th European Conference* (Vol. 4648, p. 233-242). Berlin: Springer.

Berg, H. C. (2004). *E. coli in motion.* New York: Springer.

Berg, H. C., & Brown, D. A. (1972, 27 October). Chemotaxis in *Escherichia coli* analysed by three-dimensional tracking. *Nature, 239*, 500-504.

Blount, Z. D., Borland, C. Z., & Lenski, R. E. (2008, June). Historical contingency and the evolution of a key innovation in an experimental population of *Escherichia coli. Proceedings of the National Academy of Science, 105*(23), 7899–7906.

Blynel, J., & Floreano, D. (2003). Exploring the T-maze: evolving learning-like robot behaviors using CTRNNs. In S. Cagnoni (Ed.), *Applications of evolutionary computing* (pp. 593–604). Berlin: Springer.

Braitenberg, V. (1986). *Vehicles: experiments in synthetic psychology.* Cambridge, MA: MIT Press.

Briggman, K. L., Arbanel, H. D. I., & Kristan, Jr., W. B. (2005). Optical imaging of neural populations during decision-making. *Science, 307*(5711), 896-901.

Carlile, M. J. (1975). Taxes and tropisms: diversity, biological significance and evolution. In M. J. Carlile (Ed.), *Primitive sensory and communication systems: the taxes and tropisms of micro-organisms and cells* (p. 1-28). London: Academic Press.

Clune, J., Ofria, C., & Pennock, R. T. (2007). Investigating the emergence of phenotypic plasticity in evolving digital organisms. In F. Almeida e Costa, L. Rocha, E. Costa, I. Harvey, & A. Coutiho (Eds.), *Advances in Artificial Life* (pp. 74–83). Berlin: Springer.

Collett, T. S. (1993). Route following and the retrieval of memories in insects. *Comparative Biochemistry and Physiology Part A, 104A*, 709–716.

Collett, T. S., & Baron, J. (1995). Learnt sensori-motor mappings in honeybees: interpolation and its possible relevance to navigation. *Journal of Comparative Physiology A, 177*, 287–298.

Collett, T. S., & Collett, M. (2002). Memory use in insect visual navigation. *Nature Reviews Neuroscience, 3*, 542–552.

Collett, T. S., Collett, M., & Wehner, R. (2001). The guidance of desert ants by extended landmarks. *Journal of Experimental Biology, 204*, 1635-1639.

Collett, T. S., Fry, S. N., & Wehner, R. (1993). Sequence learning by honeybees. *Journal of Comparative Physiology A, 172*, 693–706.

Collett, T. S., & Graham, P. (2004). Animal navigation: path integration, visual landmarks, and cognitive maps. *Current Biology, 14*, R475-R477.

Collett, T. S., Graham, P., & Durier, V. (2003). Route learning by insects. *Current Opinion in Neurobiology, 13*, 718–725.

Conway Morris, S. (2003). *Life's solution.* Cambridge, UK: Cambridge University Press.

De Jong, K. A. (2006). *Evolutionary computation: a unified approach.* Cambridge, MA: MIT Press.

Dennett, D. C. (2002). The new replicators. In M. Pagel (Ed.), *Encyclopedia of Evolution* (pp. E83–E92). New York: Oxford University Press.

Dickinson, J. A. (1994). Bees link local landmarks with celestial compass cues. *Naturwissenschaften, 81*, 465-467.

Dukas, R. (1998). Evolutionary ecology of learning. In R. Dukas (Ed.), *Cognitive ecology: the evolutionary ecology of information processing and decision making* (pp. 129–174). Chicago: University of Chicago Press.

Dukas, R. (1999). Costs of memory: ideas and predictions. *Journal of Theoretical Biology, 197*, 41–50.

Dukas, R. (2008). Evolutionary biology of insect learning. *Annual Review: Entomology, 53*, 145–160.

Dukas, R., & Bernays, E. A. (2000). Learning improves growth rate in grasshoppers. *Proceedings of the National Academy of Science, 97*(6), 2637–2640.

Dukas, R., & Duan, J. J. (2000). Potential fitness consequences of associative learning in a parasitoid wasp. *Behavioral Ecology, 11*(5), 536–543.

Dyer, F. C. (1991). Bees acquire route-based memories but not cognitive maps in a familiar landscape. *Animal Behaviour, 41*, 239-246.

Dyer, F. C. (1998). Cognitive ecology of navigation. In R. Dukas (Ed.), *Cognitive ecology: the evolutionary ecology of information processing and decision making* (p. 201-260). Chicago: University of Chicago Press.

Dyer, F. C., & Dickinson, J. A. (1994). Development of sun compensation by honeybees: how partially experienced bees estimate the sun's course. *Proceedings of the National Academy of Sciences of the United States of America, 91*, 4471-4474.

Dyer, F. C., & Dickinson, J. A. (1996). Sun-compass learning in insects: representation in a simple mind. *Current Directions in Psychological Science, 5*(3), 67-72.

Dyer, F. C., & Gould, J. L. (1981). Honey bee navigation: a backup system for cloudy days. *Science, 214*(4524), 1041-1042.

Eichenbaum, H. (2008, June). Memory. *Scholarpedia, 3*(3), 1747.

Elsberry, W. R., Grabowski, L. M., & Pennock, R. T. (2009, March 31–April 2). Cockroaches, drunkards, and climbers: modeling the evolution of simple movement strategies using digital organisms. In *Proceedings of the 2009 IEEE Symposium on Artificial Life (IEEE ALife 2009).* Nashville, TN: IEEE.

Floreano, D. (1997). Ago ergo sum. In G. Mulhauser (Ed.), *Evolving consciousness.* Amsterdam: J. Benjamins.

Floreano, D., Mondada, F., Perez-Uribe, A., & Roggen, D. (2004). Machine self-evolution. *YLEM Journal, 24*(12), 4–10.

Floreano, D., & Urzelai, J. (2000). Evolutionary robots with online self-organization and behavioral fitness. *Neural Networks, 13*, 431–443.

Gallistel, C. R. (1990). *The organization of learning.* Cambridge, MA: MIT Press.

Godfrey-Smith, P. (2002). Environmental complexity and the evolution of cognition. In R. J. Sternberg & J. C. Kaufman (Eds.), *The evolution of intelligence* (pp. 223–249). Mahwah, New Jersey: Lawrence Erlbaum Associates.

Goldsby, H., Knoester, D., Cheng, B., McKinley, P., & Ofria, C. (2007, May). Digitally evolving models for dynamically adaptive systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 2007 (SEAMS '07)* (pp. 13–13). Los Alamitos, CA: IEEE Computer Society Press.

Gould, S. J. (1989). *Wonderful life: the Burgess Shale and the nature of history.* New York: W. W. Norton.

Grabowski, L. M., Elsberry, W. R., Ofria, C., & Pennock, R. T. (2008). On the evolution of motility and intelligent tactic response. In *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (pp. 209–216). New York, NY, USA: ACM.

Graham, P., Fauria, K., & Collett, T. S. (2003). The influence of beacon-aiming on the routes of wood ants. *Journal of Experimental Biology, 206*, 535–541.

Hartmann, G., & Wehner, R. (1995). The ant's path integration system: a neural architecture. *Biological Cybernetics, 73*, 483-497.

Kacelnik, A., & Krebs, J. R. (1985). Learning to exploit patchily distributed food. In R. M. Sibly & R. H. Smith (Eds.), *Behavioral ecology* (pp. 189–205). Oxford: Blackwell.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Knoester, D. B., McKinley, P. K., & Ofria, C. (2007, July). Using group selection to evolve leadership in populations of self-replicating digital organisms. In *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (pp. 293–300). New York: ACM.

Kohonen, T. (1977). *Associative memory: a system-theoretical approach.* Berlin: Springer-Verlag.

Kohonen, T. (2001). *Self-organizing maps* (3rd ed.). Berlin: Springer.

Koshland, D. E. (1979, October). A model regulatory system: bacterial chemotaxis. *Physiological Reviews, 59*(4), 811–862.

Kotsiantis, S. B. (2007). Supervised machine learning: a review of classification techniques. *Informatica, 31*, 249–268.

Kotsiantis, S. B., & Pintelas, P. (2004). Recent advances in clustering: a brief survey. *WSEAS Transactions on Information Science and Applications, 1*(1), 73–81.

Lehrer, M. (1991, June). Bees which turn back and look. *Naturwissenschaften, 78*(6), 274–276.

Lenski, R. E., Ofria, C., Pennock, R. T., & Adami, C. (2003). The evolutionary origin of complex features. *Nature, 423*, 139–144.

Mayley, G. (1996). The evolutionary cost of learning. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, & S. W. Wilson (Eds.), *From Animals to Animats 4: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior* (pp. 458–467). Cambridge, MA: The MIT Press.

Maynard Smith, J. (1992, 27 February). Byte-sized evolution. *Nature, 355*, 772-773.

Maynard Smith, J. (1986). *The problems of biology.* Oxford: Oxford University Press.

Mery, F., & Kawecki, T. J. (2002). Experimental evolution of learning ability in fruit flies. *Proceedings of the National Academy of Science, 99*(22), 14272–14279.

Mery, F., & Kawecki, T. J. (2003). A fitness cost of learning ability in *Drosophila melanogaster. Proceedings of the Royal Scociety B, 270*, 2465–2469.

Mery, F., & Kawecki, T. J. (2004). An operating cost of learning in *Drosophila melanogaster. Animal Behaviour, 68*, 589–598.

Mery, F., & Kawecki, T. J. (2005, May). A cost of long-term memory in *Drosophila. Science, 308*, 1148.

Mittelstaedt, H. (1985). Analytical cybernetics of spider navigation. In F. G. Barth (Ed.), *Neurobiology of arachnids* (p. 298-316). Berlin: Springer.

Müller, M., & Wehner, R. (1988). Path integration in desert ants, *Cataglyphis fortis. Proceedings of the National Academy of Sciences of the United States of America, 85*, 5287-5290.

Nakagaki, T., Yamanda, H., & Toth, Á. (2000, 28 September). Maze-solving by an amoeboid organism. *Nature, 407*, 470.

Newell, A., & Simon, H. A. (1956). The logic theory machine: a complex information processing system. *IEEE Transactions on Information Theory, 2*(3), 61–79.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: the biology, intelligence, and technology of self-organizing machines.* Cambridge, MA: MIT Press.

Nolfi, S., & Floreano, D. (2002). Synthesis of autonomous robots through evolution. *Trends in Cognitive Sciences, 6*(1), 31–37.

Nolfi, S., Miglino, O., & Parisi, D. (1994). Phenotypic plasticity in evolving neural networks. In *Proceedings of the PerAc '94 Conference* (pp. 146–157). Los Alamitos, CA: IEEE Computer Society Press.

Ofria, C., Adami, C., & Collier, T. C. (2002). Design of evolvable computer languages. *IEEE Transactions in Evolutionary Computation, 17*, 528-532.

Ofria, C., & Wilke, C. O. (2004). Avida: a software platform for research in computational evolutionary biology. *Artificial Life, 10*(2), 191–229.

Ordal, G. W. (1980). Bacterial chemotaxis: a primitive sensory system. *BioScience, 30*(6), 408–411.

Pennock, R. T. (2007). Models, simulations, instantiations, and evidence: the case of digital evolution. *Journal of Experimental and Theoretical Artificial Intelligence, 19*(1), 29–42.

Rescorla, R. A. (1988). Behavioral studies of Pavlovian conditioning. *Annual Review of Neuroscience, 11*, 329–352.

Roche, R. A. P., Mangaong, M. A., Commins, S., & O'Mara, S. M. (2005). Hippocampal contributions to neurocognitive mapping in humans: a new model. *Hippocampus, 15*, 622-641.

Shettleworth, S. J. (1998). *Cognition, evolution, and behavior.* New York: Oxford University Press.

Simon, H. A. (1957). *Models of man—social and rational.* New York: John Wiley and Sons.

Stanley, K., Bryant, B., & Miikkulainen, R. (2003). Evolving adaptive neural networks with and without adaptive synapses. In *2003 Congress on evolutionary Computation (CEC '03* (Vol. 4, pp. 2557–2564). Los Alamitos, CA: IEEE Computer Society Press.

Stephens, D. W. (1991). Change, regularity, and value in the evolution of animal learning. *Behavioral Ecology, 2*, 77–89.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction.* Cambridge, MA: MIT Press.

Tinbergen, N. (1951). *The study of instinct.* Oxford: Clarendon Press.

Todd, P. M., & Miller, G. F. (1990). Exploring adaptive agency II: simulating the evolution of associative learning. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior on* (pp. 306–315). Cambridge, MA, USA: MIT Press.

Trussell, G. C., & Smth, L. D. (2000). Induced defenses in response to an invading crab predator: an explanation of historical and geographic phenotypic change. *Proceedings of the National Academy of Science, 29*(97), 2123–2127.

Tuci, E., Quinn, M., & Harvey, I. (2002). An evolutionary ecological approach to the study of learning behavior using a robot-based model. *Adaptive Behavior, 10*, 201–221.

von Frisch, K. (1967). *The dance language and orientation of bees.* Cambridge, MA: Harvard University Press.

Waddington, C. H. (1942). Canalization of development and the inheritance of acquired characters. *Nature, 150*, 563–565.

Wehner, R., Michel, B., & Antonsen, P. (1996). Visual navigation in insects: coupling of egocentric and geocentric information. *Journal of Experimental Biology, 199*, 129-140.

Wei, C. A., Rafalko, S. L., & Dyer, F. C. (2002). Deciding to learn: modulation of learning flights in honeybees, *Apis mellifera. Journal of Comparative Physiology A, 188*, 725–737.

Yamauchi, B., & Beer, R. D. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In D. Cliff, P. Husbands, J. A. Meyer, & S. W. Wilson (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 382–391). Cambridge, MA: MIT Press.

Zhang, S. W., Bartsch, K., & Srinivasan, M. V. (1996). Maze learning by honeybees. *Neurobiology of Learning and Memory, 66*, 267–282.

Zhang, S. W., Lehrer, M., & Srinivasan, M. V. (1999). Honeybee memory: navigation by associative grouping and recall of visual stimuli. *Neurobiology of Learning and Memory, 72*, 180–201.

Zhang, S. W., Mizutani, A., & Srinivasan, M. V. (2000). Maze navigation by honeybees: learning path regularity. *Learning and Memory, 7*, 363–374.