Portland State University

PDXScholar

Dissertations and Theses Dissertations and Theses

Winter 2-10-2015

Chemical Reaction Network Control Systems for Agent-Based Foraging Tasks

Joshua Stephen Moles
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Biomedical Commons, and the Other Electrical and Computer Engineering Commons
Let us know how access to this document benefits you.

Recommended Citation
Moles, Joshua Stephen, "Chemical Reaction Network Control Systems for Agent-Based Foraging Tasks" (2015). *Dissertations and Theses.* Paper 2203.
https://doi.org/10.15760/etd.2200

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Chemical Reaction Network Control Systems for Agent-Based Foraging Tasks

by

Joshua Stephen Moles

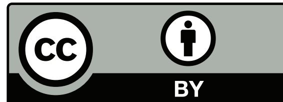
A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Christof Teuscher, Chair
Marek A. Perkowski
Eric Wan

Portland State University
2015

© 2014 Joshua Stephen Moles



This work is licensed under the Creative Commons Attribution 4.0

International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by/4.0/.

Abstract

Chemical reaction networks are an unconventional computing medium that could benefit from the ability to form basic control systems. In this work, we demonstrate the functionality of a chemical control system by evaluating classic genetic algorithm problems: Koza's Santa Fe trail, Jefferson's John Muir trail, and three Santa Fe trail segments. Both Jefferson and Koza found that memory, such as a recurrent neural network or memories in a genetic program, are required to solve the task. Our approach presents the first instance of a chemical system acting as a control system. We propose a delay line connected with an artificial neural network in a chemical reaction network to determine the artificial ant's moves.

We first search for the minimal required delay line size connected to a feed forward neural network in a chemical system. Our experiments show a delay line of length four is sufficient. Next, we used these findings to implement a chemical reaction network with a length four delay line and an artificial neural network. We use genetic algorithms to find an optimal set of weights for the artificial neural network. This chemical system is capable of consuming 100% of the food on a subset and greater than 44% of the food on Koza's Santa Fe trail.

We also show the first implementation of a simulated chemical memory in two different models that can reliably capture and store information over time. The ability to store data over time gives rise to basic control systems that can perform more complex tasks. The integration of a memory storage unit and a control system in a chemistry has applications in biomedicine, like smart drug delivery. We show that we can successfully store the information over time and use it to act as a memory for a control system navigating an agent through a maze.

Acknowledgements

I would like to take a moment and extend a special thank you to all of the individuals who have helped me through the journey to complete this thesis. I first would like to thank my adviser, Dr. Christof Teuscher, for all of his guidance throughout my time at Portland State University. Through the ups and downs of this process, Dr. Teuscher always encouraged me to stick with it. I would also like to thank the committee members, Dr. Eric Wan and Dr. Marek Perkowski, for their advice and time on this thesis. I would like to thank Prof. Mark Faust for the encouragement to opt for the thesis route instead of coursework as well as general advice throughout my career at Portland State. I also extend a thank you to my lab colleague, Peter Banda, that provided me with the help to ramp up on this topic and work with me on publishing my first paper.

I am grateful to my family who encouraged me to complete this program. I appreciate the advice provided throughout this degree by members of my family: Michelle Moles, Steve Moles, Courtney Moles, and Scott Moles. I especially would like to thank my friend, Jacob Couch, for his advice, encouragement, patience, and proofreading. I also would like to acknowledge my friends who have helped me throughout this research and coursework at Portland State. I also appreciate the patience of these outstanding individuals while I completed this thesis while working full time. This includes, and is not limited to Kurt Kolkind, Gabriel Thompson, Daniel Lenski, Brian Jones, Giedre Novikaite, Aaron Anderson, Erik Rhodes, Lindsey Geer, Anne Van Hulle, Eric Krause, and Scott Cline.

Contents

Abstract i

Acknowledgements ii

List of Tables vi

List of Figures vii

Acronyms x

1 Introduction 1

1.1 Objectives . . . 2

1.2 Approach . . . 3

1.3 Significance . . . 4

1.4 Structure . . . 6

2 Background 8

2.1 Trail Problems . . . 8

2.2 Perceptrons and Artificial Neural Networks . . . 14

2.3 Genetic Algorithms . . . 18

2.4 Chemical Reaction Networks . . . 21

3 Delay Line 26

3.1 Delay Line Concept . . . 26

3.2 Delay Line Design . . . 27

3.2.1 Manual Copy Delay Line 27
3.2.2 Backwards Signal Propagation Delay Line 30
3.2.3 Inherit Single Instruction, Multiple Data 32
3.2.4 More than Two Stages 33
3.3 Results 34
3.3.1 Two Stages 35
3.3.2 Over Two Stages 37
3.3.3 Time Series Perceptron Integration 38
3.4 Discussion 39

4 Trail Runner and Trail Viewer 50

4.1 Trail Runner 50
4.2 Trail Viewer 52

5 Non-Chemical Reaction Network Simulations 55

5.1 Methodology 55
5.2 Results 61
5.3 Discussion 77

6 Chemical Reaction Network Trail Simulations 80

6.1 Methodology 80
6.2 Results 85
6.3 Discussion 98

7 Chemical Reaction Network Realization 103

7.1 Chemical Representation 103
7.2 Processing Speed 106

8 Conclusion **109**

8.1 Contributions 110

8.2 Future Work 111

References **113**

List of Tables

3.1 Delay Line Values Over Time 27

3.2 Two Stage Manual Delay Line Actions 30

3.3 Pipeline View of Data in Two Stage Manual Delay Line 30

3.4 Two Stage Backwards Propagation Delay Line 32

3.5 Two Stage Manual Copy Delay Line Rate Constants 35

3.6 Two Stage Backwards Propagating Delay Line Rate Constants . . . 36

3.7 Error for Both Delay Lines 37

5.1 Summary of GA Parameters for Test Runs 59

5.2 Moves Limits for Test Runs 59

6.1 AASP Reaction Set 84

6.2 Run Time Benchmark of COEL Generations 87

6.3 Summary of Results 93

List of Figures

1.1 Approach Process 4

2.1 John Muir Trail 10

2.2 Jefferson John Muir Trail ANN 11

2.3 Santa Fe Trail 12

2.4 Diagram of Three Input Perceptron 15

2.5 Example Artificial Neural Network 16

2.6 Bäck Simple Genetic Algorithm 22

3.1 Delay Line 26

3.2 Two Stage Manual Copy Delay Line 28

3.3 Two Stage Manual Copy Delay Line I/O 41

3.4 Two Stage Manual Copy Delay Line Signaling 42

3.5 Two Stage Backwards Propagating Delay Line 43

3.6 Two Stage Backwards Propagating Delay Line I/O 44

3.7 Two Stage Backwards Propagating Delay Line Signaling 45

3.8 Single Instruction, Multiple Data Representation with Delay Line . 46

3.9 Three Stage Backwards Propagating Delay Line 46

3.10 Error for Both Delay Lines 47

3.11 Perceptron Integration with Backwards Propagating Delay Line . . 47

3.12 Success Rate with Delay Line and Perceptron Integration 48

3.13 Concentration Traces for Perceptron/Delay Line Learning Logic OR 49

4.1 Two Test Trails 52

4.2 Screen Shot of Trail Viewer 54

5.1 Genetic Algorithm Flow Chart 56

5.2 Three Santa Fe Trail Segments 60

5.3 Delay Line with ANN for Trails 61

5.4 Food Consumed in Test Trail 1 62

5.5 Individual Path in Test Trail 1 63

5.6 Food Consumed in Test Trail 2 64

5.7 Individual Path in Test Trail 2 65

5.8 Food Consumed in John Muir Trail 66

5.9 Individual Path in John Muir Trail 67

5.10 Food Consumed in John Muir Trail on Stuck Run 68

5.11 Run Time Benchmark of Trail Runner 69

5.12 Moves Type in Santa Fe Trail 70

5.13 Maximum Food Gathered by Varying Delay Line Length 71

5.14 Mean Food Gathered on Easy Segment with Varying Delay Line Length 72

5.15 Mean Food Gathered on Medium Segment with Varying Delay Line Length 73

5.16 Mean Food Gathered on Hard Segment with Varying Delay Line Length 74

5.17 Mean Food Gathered on Santa Fe Trail with Varying Delay Line Length 75

5.18 Mean Food Gathered on All Segments with Varying Delay Line Length 76

6.1 Delay Line with ANN in Chemistry 81

6.2 Chemical Implementation with AASP 82
6.3 COEL Run Time Benchmark 85
6.4 Run Time Benchmark of COEL Generations 86
6.5 Maximum Normalized Food Results with CRN 88
6.6 Mean Normalized Food Results with CRN 89
6.7 Probability of Food on Trail 90
6.8 CRN Percent Error Against Maximum Available 91
6.9 CRN Percent Error Against Non-CRN 92
6.10 Best CRN with Hidden Individual Evaluated on Other Trails 94
6.11 Best Individual No Hidden Evaluated on Other Trails 95
6.12 Histogram of Food with Hidden Layer 96
6.13 Histogram of Food with No Hidden Layer 97

7.1 Example of Deoxyribozyme Implementation 105

Acronyms

AASP Analog Asymmetric Signal Perceptron. 24, 81, 82, 105

AC Artificial Chemistry. 2, 21, 23

ANN Artificial Neural Network. 1, 2, 3, 6, 8, 9, 11, 9, 11, 13, 14, 16, 17, 21, 24, 25, 39, 50, 51, 55, 58, 59, 60, 61, 60, 70, 77, 80, 81, 82, 88, 94, 98, 100, 103, 105, 107, 109, 110, 111

ASP Asymmetric Signal Perceptron. 24, 26, 39, 109, 110

BPL Backwards signal Propagation delay Line. 30, 31, 33, 35, 39, 33, 82, 109, 110, 112

CMOS Complementary Metal-Oxide-Semiconductor. 1, 103

COEL COllective cELlular computing. 24, 51, 53, 80, 85

CRN Chemical Reaction Network. 2, 3, 5, 6, 8, 21, 24, 25, 26, 39, 50, 51, 53, 55, 77, 78, 80, 81, 82, 85, 88, 91, 94, 96, 98, 99, 100, 102, 106, 109, 110, 111, 112

DEAP Distributed Evolutionary Algorithms in Python. 51, 53, 55

DNA DeoxyriboNucleic Acid. 103, 105, 106

FIFO First-In, First-Out. 1, 26

FSM Finite State Machine. 9, 11, 13, 20

GA Genetic Algorithm. 8, 9, 15, 17, 18, 19, 20, 21, 24, 26, 33, 50, 51, 52, 51, 52, 53, 55, 58, 60, 62, 66, 68, 69, 71, 72, 77, 80, 82, 85, 96, 98, 102, 109, 110, 111

GGTCA Glycolysis/Gucoeogensis mitocondrial TiCarboxylic Acid. 103

GP Genetic Programming. 11, 13

GPN Genetically Programmed Network. 13

GUI Graphical User Interface. 53

HPC High-Performance Computing. 24

LFSR Linear Feedback Shift Register. 3, 39, 112

MDL Manual copy Delay Line. 26, 28, 29, 30, 32, 33, 35, 37, 39, 30, 71, 81, 82, 85, 88, 104, 105, 107, 109, 110, 111, 112

MOEA MultiObjective Evolutionary Algorithms. 20

SCOOP Scalable COncurrent Operations in Python. 51, 53, 69, 77

SIMD Single Instruction, Multiple Data. 32

SMAPE Symmetric Mean Absolute Percentage Error. 35, 36, 37, 39, 37

SSD Solid State Drive. 69

VPS Virtual Private Server. 69

Chapter 1

Introduction

Chemistry as an alternative computer paradigm provides a means to perform decision making in areas that conventional systems are unable to operate. As an example, Complementary Metal-Oxide-Semiconductor (CMOS) is fairly impractical for use in a wet system at a biological cellular level. At present, chemistry lacks a means to represent some of the more mature models found in systems like CMOS. Many of these components are in their infancy. Chemistries also provide an interesting alternative computing means with their natural parallelism because all reactions and changes in concentration of species occur concurrently [1].

Developing and demonstrating the application of some of these seemingly simple blocks are fundamental to build more complex systems. As an example, memory storage is a fundamental building block for calculation and processing [2]. Retrieving previous results or observations are necessary to build more complex control systems and devices. Once we have memories, it opens the door to implementing systems capable of more complicated processing. Memory is a necessary building block to store data for processing and storing things like data or instructions for operations. In addition, memory is a useful block when constructing automata. Arkin and Ross emphasized the need for "buffer" between the phases of the Boolean logic elements they construct [3]. A data structure, such as a memory, could meet the buffer Arkin and Ross call for.

Decision making ability is just as crucial as memory when building larger systems. Modeling decision making with neuron models connected together to form

systems known as Artificial Neural Network (ANN), leads to greater system complexity. Chemistry could benefit from more models of memories, calculation blocks, and control systems to demonstrate the complex decision making capability of the medium. The demonstration of a system like we are proposing is essential to form systems capable of more complex tasks such as dynamic length memories, First-In, First-Outs (FIFOs), networking protocols [4], logic circuits [3] [5] [6] [7], arithmetic [8], signal processing [9], or games [10] [11].

1.1 Objectives

This thesis is meant to perform an investigation of the use of chemistry to solve a control system problem. At present, there are few examples of implementing control systems as chemistries. The control system problem implemented is the ant trail task originally presented by Jefferson to solve the John Muir trail [12].

In this task a control system, represented as an ant, must navigate through a trail consuming as many pieces of food as possible with 200 moves. The food elements starts out one after another, but quickly gets more difficult as the trail continues on by adding turns and gaps. Jefferson's original work on this subject found that memory is necessary to solve this task, such as recurrence in a ANN. Since this type of recurrence is presently unimplemented in chemistry, we used memories. This requires figuring out the needed size of the memory as well as what type of network best performs the actual computation. The objective is to use a series of reactions and species that model a real world chemistry in a simulated, computer environment to represent the ant's decisions. This set of reactions and species is also known as an Chemical Reaction Network (CRN), which is an instance of an Artificial Chemistry (AC). We will discuss CRNs later

in the next chapter. Such a system requires the ability to store information over time and is more complex than present systems implemented a chemistry.

The first step is the chemistry implementation of a memory. Our goal is to have a minimal length memory that can accurately capture the information for later access in navigating the trail. This gives us the ability to perform random memory access as well as form more complex data structures in a chemistry environment. We will show two different models that trade off complexity for accuracy. One provides greater storage length requiring the user to manually signal every movement of data within the delay line. The other provides a limited storage length with less user intervention to signal the transition of the values.

1.2 Approach

Our approach was to first look at the necessary elements to solve the problem. Figure 1.1 shows our process described here. Based off the work by Jefferson and Koza, we found that we required a memory and the ability to represent ANNs and perceptrons in a chemistry. Without models of memory in a chemistry, we first designed the chemical delay lines to store previous trail information. Next, an evaluation outside of the chemistry is done to determine the minimal network layout and memory size for a chemical implementation. The optimization in the non-chemical environment is an essential step due to the drastically larger simulation time when moving into a chemistry simulator. In addition, the ANN that Jefferson directly uses would require a significant amount of time to simulate so a study to simplify the system is required. Finally, taking this layout and memory, we use existing chemical perceptron models that are modeled with a set of reactions and species [13] [14] to simulate the system in a chemical environment.

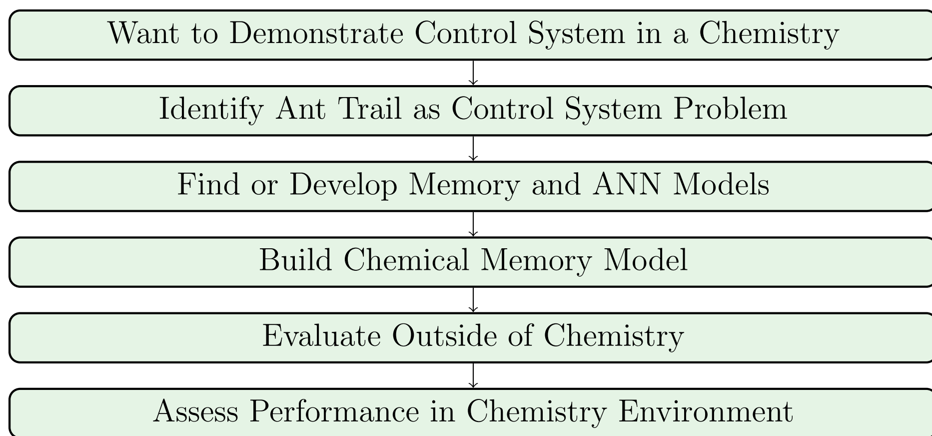


Figure 1.1: Chart showing approach we took to address problem. First, we seek implementation of a control system, as a CRN. We identify the ant trail problem as one to approach. Then, we find or construct the blocks needed for our system. We identify existing ANN models. We do not find a chemical memory, so we develop that block. Then, determine the best performance outside of a chemistry before testing the system in an artificial chemistry environment.

1.3 Significance

The work presented here shows for the first time that data has been stored in a directed fashion within a chemistry for later processing. This delay line created here is a building block to larger control systems. This is exemplified by connecting the delay line to an artificial neural network composed of chemical perceptrons [7] that are capable of finding solutions to the ant trail problems. Combining the delay line with a perceptron in a system like the trail solving shows how we can take two modular systems, connect them together, and create more complex agent-based systems in chemistries. As an example, a delay line paired with an XOR allows construction of systems like a Linear Feedback Shift Register (LFSR). Fields like signal processing, networking, smart medication delivery, and harmful bacteria detection all could benefit from a chemistry-based memory.

An autonomous agent capable of making control decisions is a building block for

larger, more complex systems [15]. Demonstration of the delay line in combination with a system like the trail problem allows problems that were once unsolvable in a chemistry are now implementable. In addition, the construction of the delay line independently of the construction of the chemical perceptron [7] shows how the blocks are added or removed to build a more complex system.

Others have implemented systems in CRNs that act similar to a buffer or memory. Jiang *et al.* introduced the concept of a delay element [9]. The delay element is primarily used as a storage area for holding data in between each computation cycle. The data then returns and is examined in computing during the next iteration of the calculation. Jiang's buffer is primarily a signal processing application looking only at the previous value. Our delay line has the ability to delay not only multiple steps in time, but also allows access to any of the past values besides the most recent. We could create a FIFO [2] out of the delay line by removing the intermediate output values and providing only the final output.

Other areas, such as networking, use chemical reaction networks as a mechanism to control scheduling and queuing of packets [4]. The work discusses a methodology to use the law of mass action as a means to schedule packets. Meyer's work did not actually implement the data structure for the packets in a chemistry, but only the control with the memory stored outside of a chemical system. With a buffer like the one we are describing, then Meyer's systems could also be extended to actually implement a means to queue packets in a chemical environment. This method would reduce cost and complexity by having a single implementation medium. The available of a memory in chemistry would be helpful to address several potential applications.

One such example in the field of biomedicine is smart medication like drug delivery [16], injury assesment [17], “sense-act-treat” systems [18], or others [19] [20]. For drug delivery, rather than have a fixed dosage of a specific type of medicine, a patient could be observed over a time window and then adapt the drug (in quantity or species) to best respond to their needs [21] [22]. Another use in the biochemistry field would be the detection of harmful species, e.g., chemicals produced by cancer cells in a host. With a time delay line, the detection would not be limited to a simple yes or no, but can get extended to measure a chemical concentration as well as capture at what point the event occurred. Combination of the delay line with a control system, like the ant trail, demonstrates a system reacting from these inputs.

The biomedical examples are not just limited to cancer or diabetes. There are numerous other types of detection that could benefit compared to the traditional methods that either require long periods of time or handling of potentially dangerous samples. Another example is a modern *Salmonella* detection system still requires the analysis of samples overnight [23]. An OR-like perceptron connected to a delay line system in a CRN could detect and react to the presence of *Salmonella* immediately. Another is the ability to monitor blood sugar levels over time with a closed-feedback system monitoring the patient and adjust the dosage of delivered insulin [19].

1.4 Structure

This work is divided into 8 chapters. In chapter 2, we provide a background of CRNs, ANN, the trail problems, and previous work related to this thesis. Next, chapter 3 discusses the implementation and results of the two models of delay

line designed for this task. Then, chapter 4 covers the ANN solver applications written to test the trail problems in a non-CRN environment. We then use the ANN applications to find the optimal length delay line in Chapter 5. Chapter 6 goes over the combination of a CRN delay line and perceptron and presents those results. Next, we discuss the possibility of implementing the system as a wet chemistry in Chapter 7. The paper wraps up with some concluding remarks in Chapter 8.

Chapter 2

Background

This chapter discusses the background to several topics used throughout our work. We start by introducing the original artificial ant trail problem proposed by Jefferson and Koza. Next, we will discuss a model for biological neurons, the perceptron. The perceptron is used as the computational unit in the larger networks that perform computations to form Artificial Neural Networks (ANNs). Then, we discuss the use of Genetic Algorithms (GAs) as a means to optimize the function of preceptrons. A discussion on Chemical Reaction Networks (CRNs) concludes this background chapter.

2.1 Trail Problems

Jefferson *et al.* introduced the trail navigation problem in [12]. In this task, an artificial ant is placed in a 32×32 grid. The goal is for the ant to collect as many pieces of food on the trail in a limited number of moves. Figure 2.1 shows the original John Muir trail proposed by Jefferson. The trail is toroidal, meaning that the top row of the trail is adjacent to the bottom row and the left and right rows are also adjacent. Starting in the top-left corner facing right, the only input the ant receives is if there is food placed directly in front of the space that the ant is facing.

The ant makes a decision for the next action based off the only input of if there is food ahead. The actions that the ant can take are move forward, turn left, turn right, or do nothing. Taking any of these four actions (including turns)

counts as a move. As as an example, turning left followed by a forward more is two moves. The score of the ant is measured by the number of pieces of food it consumes within a limited number of moves, which was 200 in Jefferson's work. After an ant steps on a piece of food, that food is considered "consumed" so that it only receives credit for navigating over that part of the trail one time. The trail gets progressively more difficult adding gaps of increasing length and additional turns. Jefferson and Koza both designed an initial system and then evolved the parameters on it to search for an optimal control system.

Jefferson's work tested the decision making capability of the ant through Finite State Machines (FSMs) and ANNs. These FSMs were hand-designed initially and then later evolved to search for a better solution with a GA. Using FSMs, the authors found that a simple four state FSM could get a score of 42 in 200 moves. Adding a single state (to form a five-state FSM) allowed the FSM to achieve a score of 81 in 200 time steps. Given more time, Jefferson found it actually consumed all of the food in 314 time steps. After 100 generations, Jefferson found the "Champ-100" FSM that was capable of scoring an 89 (max score) through GAs.

Using the same GA configuration as the FSM, an ANN capable of scoring the maximum of 89 was found in generation 94 by Jefferson. The network they used was a recurrent ANN with two input perceptrons, five hidden units, and four output units. Figure 2.2 shows the recurrent ANN. The network featured two input perceptrons, a single hidden layer with five hidden perceptrons, and an output layer with four perceptrons. The neurons were fully forward connected, including a connection from the input layer to the output layer. Additionally, there is a recurrent connection from the hidden layer back to itself. The two inputs indicated if there was food ahead and the other was an inverse of the first.

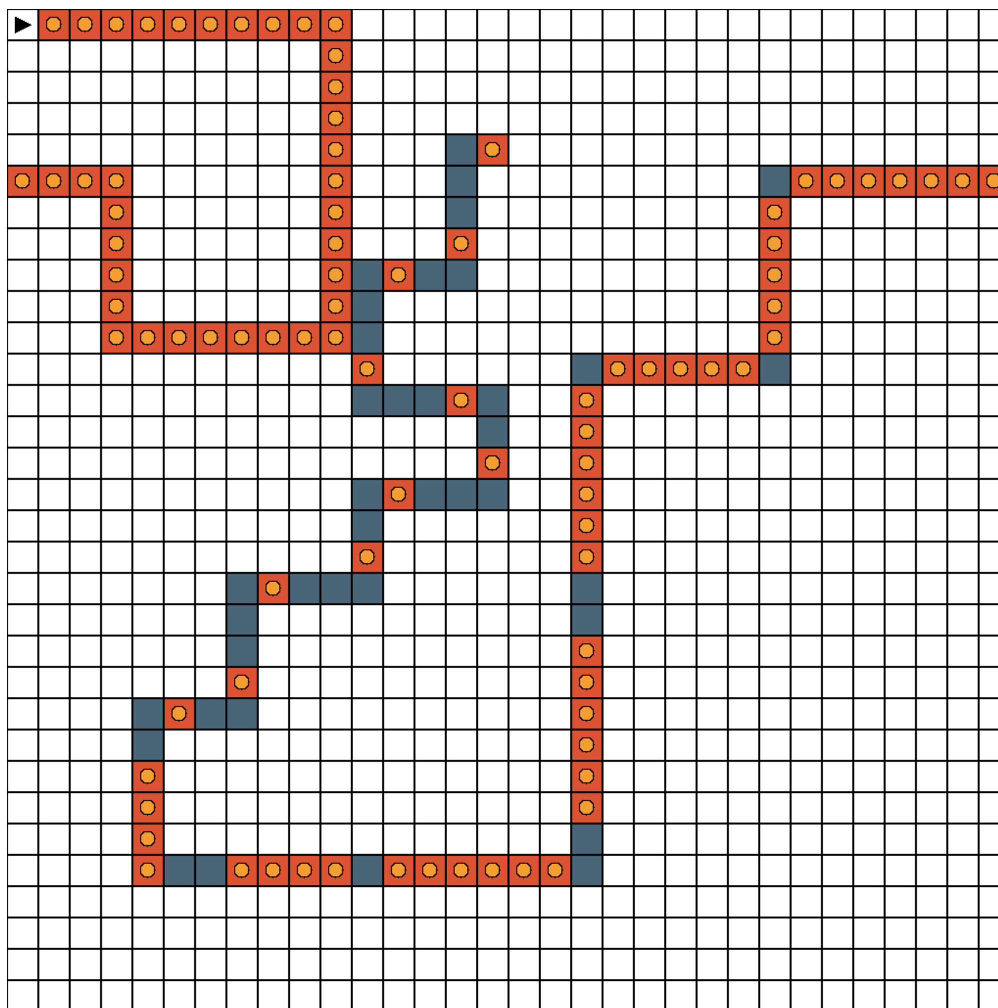


Figure 2.1: The original John Muir trail proposed by Jefferson [12]. This trail is a 32×32 toroidal grid and contains 89 pieces of food (represented by squares with circles in them). The ant is represented by an arrow in the top left corner (0, 0) facing to the right. The shaded squares that are empty are visual aids to indicate the optimal path proposed by Jefferson.

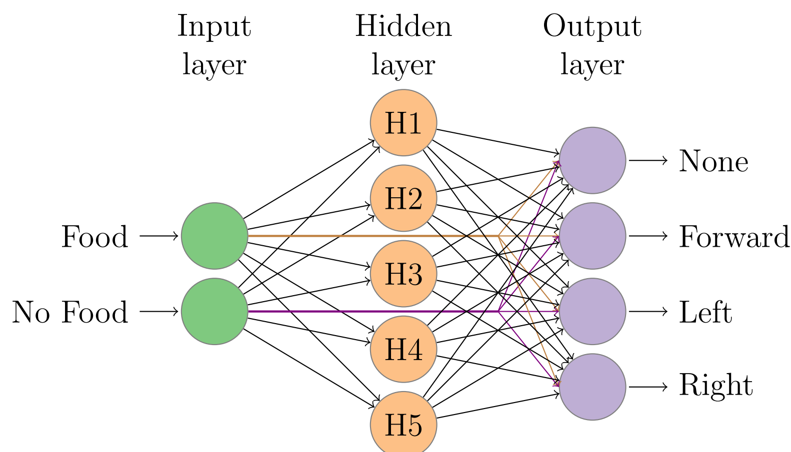


Figure 2.2: The ANN used by Jefferson in. This recurrent ANN features two input (that are the inverse of each other), five hidden, and four output units. Each of the four outputs represents an action the agent can take. The recurrent network features connections from the hidden layer back to itself (e.g., H1 back into H1) that are not shown to reduce clutter.

This is necessary since the ANN would not activate with just an input of 0 in the case of no food. The four output neurons were compared and the one with the largest output would determine the move.

Koza expanded Jefferson's work by studying the artificial ant navigating through the Santa Fe Trail [24]. According to Koza, the Santa Fe trail is a more difficult trail and is shown in Figure 2.3. The actions and task are the same as Jefferson's John Muir trail. Koza also performed analysis using evolving LISP programs instead of ANNs or FSMs like Jefferson. With the evolving LISP programs, Koza found a solution scoring the maximum (89) in generation 21.

Koza's work on the Santa Fe trail has created an entire new field of research with the optimization of evolution of these programs, also known as Genetic Programming (GP). As a result, thee Santa Fe trail tends to be a more popular trail for

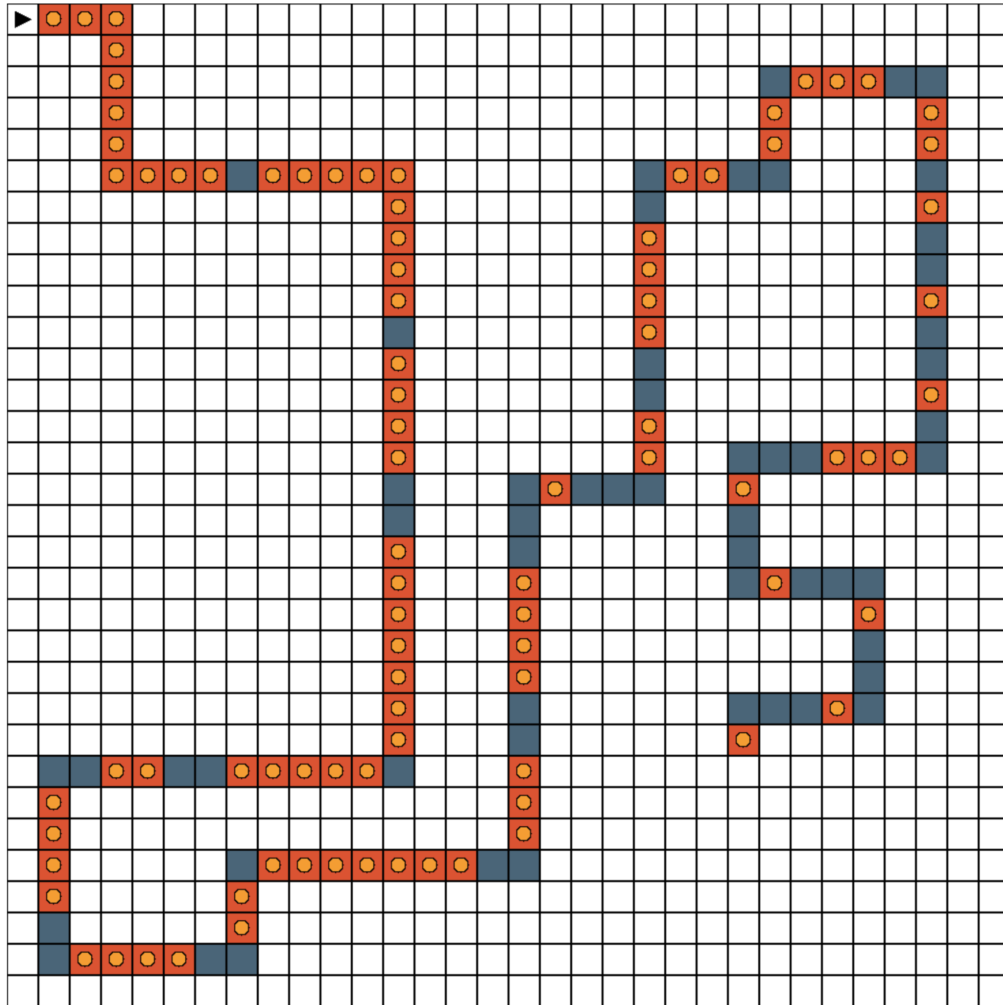


Figure 2.3: The Santa Fe Trail proposed by Koza [24]. This trail (like the John Muir Trail, Figure 2.1) is 32×32 and toroidal. The trail contains 89 pieces of food (same as trail segments in Jefferson) and are represented by squares with circles in them here. Shaded squares are provided as a visual aid to indicate the optimal route proposed by Koza. The ant is an arrow in the top left corner facing to the right.

analysis in recent research. Doucette and Heywood present an improved novelty-based fitness algorithm that they then tested against the Santa Fe trail [25]. Christensen and Oppacher presented a set of trees that efficiently search the solution space of Koza's GPs requiring less computational power [26]. The Santa Fe trail has also been used as a basis to prove implementation in reservoir computing [27].

The use of a FSM or GP seem to dominate the recent literature that we have found for directly solving the trail problem. Wilson and Kaur work on a GP representation and a modified function to improve the rate at which the system learns the task [28]. The authors develop a more effective GP to solve the problem by evaluating and improving the fitness landscape.

Chivilikhin *et al.* extend Wilson and Kaur's work with their new algorithm (MuACO*sm*) that performs well at the task of optimizing the FSM implementation [29]. Other improvements on the implementations using Koza's LISP programs were performed by Christansen and Oppacher [26] and Karmin and Ryan [30]. The algorithms for solving the trail problem we have discussed thus far are aimed more towards improving the FSM or GP implementations.

Silva *et al.* published work that discusses a hybrid combination of ANN and GP to form what they call a Genetically Programmed Network (GPN) [31]. In a GPN, the structure is laid out similar to what you would find in a ANN, but rather than have the nodes do processing that you would typically find in an ANN (like a perceptron), they are modeled by a specific program. So from an architectural layout, they follow ANNs, but the transfer function of each node is actually more akin to a GP. The authors prove functionality by consuming all pieces of food on the Santa Fe trail with a GPN.

The ANN implementation lends itself well for applications that may not have

the robust architecture or infrastructure necessary to implement such solutions. One such field is the use of chemistry to solve this problem. First, we will discuss an explanation into perceptrons and the ANN that formed Jefferson's solution to the John Muir trail.

2.2 Perceptrons and Artificial Neural Networks

McCulloch and Pitts were the pioneers of the field with their early models of neurons [32]. They presented a basic model to represent a neuron based off biological systems. Combining several of these neurons and connecting them together forms a basic ANN. Since the original work by McCulloch and Pitts, others have developed more refined models of neurons.

One such model is the perceptron presented by Rosenblatt [33] and later refined by Minsky and Papert [34]. The perceptron can act as a binary classifier. An example binary classification perceptron has multiple inputs that are transformed to one or more outputs. Each of these outputs would get converted to a binary value, such as 0 and 1, if a specified bias is met. Each of the input connections has an associated weight that determines the relative strength of that input to a different input [35]. Figure 2.4 shows a perceptron with three inputs and a single output.

The perceptron in Figure 2.4 takes three input values, x_1, x_2, and x_3 to perform a binary classification to the output, Y. Equation 2.1 shows the function that classifies a perceptron with n inputs.

$$Y = \begin{cases} 1, & \text{if } (\sum_{i=1}^{n} x_i w_i) - \theta > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

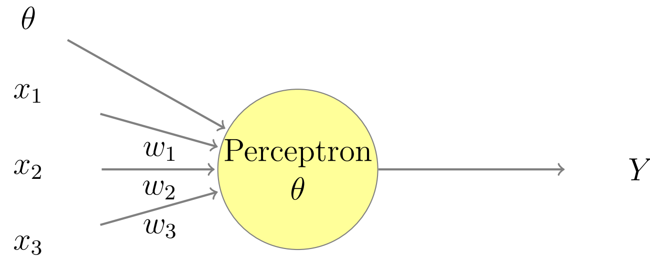


Figure 2.4: A model of a binary threshold perceptron with three inputs. This particular perceptron multiplies the weights (w_n) by each corresponding input (x_n). If the sum of these values exceeds a bias value (θ), the output (Y) is 1, otherwise, it is 0.

Perceptrons typically contain a bias, represented by θ, that allows is the adjustment for the threshold point. In other words, if the sum of the weights is not greater than the bias, then this perceptron's output will result in 0.

Perceptrons can act as more than just binary classifiers. Perceptrons can also behave in an analog fashion where the output is any range of values [33]. For example, rather than assigning the values to binary values, they could map to values on a step, sigmoid, or hyperbolic tangent function [35] to perform a desired transformation that may translate closer to the behavior of biological neurons.

The weights on the perceptron are user definable, but this is generally impractical when connecting several perceptrons together to form a larger network. One way the weights are set is by a process known as learning. In learning, the weights of the perceptron are adjusted to obtain the desired output. A basic method to learn is to randomly generate a set of test vectors and map them to the desired output [35]. Then, run each of these test vectors through the perceptron. If the desired output is obtained, no action is necessary on the weights. If the output is not the expected output, then the weights are either increased or decreased in proportion to the error in an attempt to get closer to the desired result.

An alternative means to search for the weight values is through the use of Genetic Algorithm (GA). Through the process of selecting the top performers, the weights are initially randomly set and the best performing weights are carried forward in an evolutionarily process. This type of process allows arrival at an optimal solution modeled after biological evolution rather than the user having to specify or search for values. For larger networks, this is even impractical. The weights on the networks of perceptrons later are optimized using GAs which we will discuss in the next section.

Now, taking several of the perceptrons and connecting them together forms what is referred to as an Artificial Neural Network (ANN). An ANN is a system that connected together several of these perceptrons (or another neuron model) that can adapt to a particular task and elicit a desired response. An ANN is an alternative model for computation on information [35]. They are typically formed by connecting the output of one perceptron to the input of another. This cascading of the perceptrons forms what are referred to as layers. Recall the weights on each of a perceptron's inputs allows us to control the output, so by cascading these elements together, it is possible to form complex networks of perceptrons. A typical ANN may look like that in Figure 2.5.

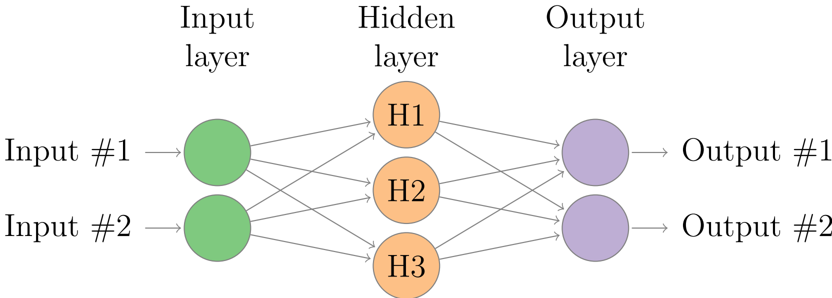


Figure 2.5: Example ANN with an input layer, single hidden layer, and output layer. Each layer has two, three, and two perceptrons, respectively.

The ANN in Figure 2.5 is composed of seven perceptrons in three layers. This example network is also known as a feedforward network where signals only propagate forward and there is effectively no tracking of state or previous values. A recurrent network is one with connections back to itself or somewhere else in the network [35]. This recurrence gives a ANN a form of memory storage of previous state information. Passing the present value back to the perceptron acts effectively as a length one memory into the previous calculation.

The hidden layer of an ANN is the levels that are between the input and output layer. Some networks may contain numerous hidden layers or none at all. The connections between the perceptrons have associated weights. For example, neuron $H1$ has one arrow coming from Input #1 and Input #2 that would each have an associated, unique weight. As the size of the network continues to grow, so does the complexity for determining the weights to appropriately accomplish a desired task. The weights of the network are adjusted by hand (for small networks), through use of a GA, or also updated in some instances with a method known as backpropagation. In short, the method calculates an error from ideal output through the neural network and updates the weights in a way to attempt to arrive at a more accurate solution [36].

The weights for the ANNs in this work are set using a GA because it was the same method that Jefferson used when optimizing the ANN in the ant trail task. They are set randomly to start and then optimized through an evolutionary process. As discussed in the previous section, Jefferson conducted a similar process to arrive at the best weights for the ANN. Now, we will discuss GAs in greater detail.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) are used to search for optimal solutions to the delay line and agent navigating through the trails. GAs were first proposed by Holland [37] [38]. In this work, Holland outlined GAs as a means to take biological adaptation and use it as a means to adapt and evolve systems in a computer. Holland's approach was more a mathematical one rather than attempting to target a specific applications for the use of GAs [39]. In the last few decades, GAs have served as a method of optimization in countless applications such as control systems [40], image enhancement [41], and neural networks [42].

A GA is a type of evolutionary computation meant to model the behavior of biological evolution through adaption [43]. Like the biological counterparts, GAs are made up of a population and have a set of functions to model behaviors to obtain variation in future populations: selection, crossover (also refereed to as recombination [39]), and mutation. By evaluating a population, producing offspring, evaluating the offspring, and repeating this cycle, a solution is derived where a population is adapted to solve our particular application.

A population is made up of individuals composed of chromosomes. Each chromosome is a set of values that provide variation in control for a system. Holland originally proposed chromosomes that are made up of bit strings; however, other work has shown that a makeup of real-valued parameters and LISP symbolic expressions are also viable representations [43]. LISP symbolic expressions are a way to represent nested or recursive list of data based off the LISP programming language. Many of the problems that Koza solves in his text are represented by LISP symbolic expressions [24]. For our work, the chromosomes are typically made up of arrays in floating point values which are functionally equivalent to a bit stream. As

an example, the weights on inputs to the perceptrons making up the trail systems are represented by an array of floating point values that cause each individual to respond differently to a set of inputs.

The evaluation of performance of a individual is known as its fitness. In biological terms, this is the tendency of an individual to reproduce in an environment [39]. In GAs, the fitness is calculated by observing the actual output of a chromosome compared to the ideal or objective output of the function [43]. For instance, in the fitness of the delay line (discussed in section 3.3), individuals are measured by taking the difference between the observed value on a delayed version of the input versus the actual copy of the input.

Selection is the process of taking individuals with the higher fitness as a basis for forming offspring. There are several methods available for selection such as

proportional selection generates offspring from individuals directly proportional to their respective fitnesses [38];

roulette wheel selection assigns a probability distribution to population where the probability of selection is proportional to the fitness and then selects, one at a time, an individual from the pool [43];

tournament selection selects a specified number of individuals randomly from the population, choose the individual with the highest fitness in this group (or tournament), and repeat as desired [44] [45];

rank-based selection selects the desired number of individuals from the entire population based only off a ranking of their fitness [46];

(μ, λ) selection form an offspring basis by generating individuals for each member of the population through mutation and/or crossover and save a defined

set of them for subsequent crossover [47];

$(\mu + \lambda)$ selection similar to **$(\mu+\lambda)$** selection, but select the top performers from both the pool of offspring **and** population [47].

Combining a method with elements of rank-based selection (showing preference towards higher fitnesses) adds elitism. Elitism is a means to carry forward members with a high fitness either directly or as a basis to form a new set of offspring. Selection methods that combine more than one of these means are known as MultiObjective Evolutionary Algorithmss (MOEAs). An example of such an algorithm is NSGA-II by Deb *et al.* [48]. NSGA-II builds an algorithm similar to $(\mu + \lambda)$ selection, but uses tournament selection in addition to determine the best individuals for later formation of the offspring.

Crossover is where individuals are combined to produce offspring. A couple of common means to perform crossover are n-point crossover and uniform crossover. An n-point crossover effectively splits the parents at n points where genetic material is alternated to the two offspring from the two parents [49]. While n only must be greater than one, one is rarely used in practice where $n = 2$ is the commonly selected value [43]. The uniform crossover [50] [51] traverses bit by bit down a parent and probabilistically determines if a crossover will occur at the current position. A $p_x = 0.5$ is a commonly selected value for uniform crossovers [43] [50].

The last GA operation we will discuss here is mutation. Like selection and crossover, there are numerous works on just the discussion of mutation and its importance in GAs. There are algorithms that utilize FSMs [43], parse trees [52], or k-opt from travelling salesman problem [53]. Throughout this work, the basic mutation utilized is a basic mutation randomly selecting bits in an individual to get flipped [39]. As a small example, you could have an individual represented by

0001 that after mutation is 1001 where the first bit was mutated.

There are a few ways that all of these steps get ordered to form a GA. Also, varying the probability of each of the actions to occur will have an impact on how rapidly the population evolves. An example GA, proposed by Bäck [43], may

1. initialize a population of individual chromosomes,
2. evaluate the population to calculate the fitness of the individuals,
3. perform crossover,
4. mutate the population,
5. evaluate fitness,
6. select next members of population,
7. repeat from step 3 until desired number of generations are evaluated.

Figure 2.6 shows a flow chart of this particular GA. Later on, we will discuss the genetic algorithms used to optimize the delay line and trail systems in section 5.1. Now, with an understanding of the trail problem, how it was solved as an ANN, and how it was optimized with a GA, we now shift focus to discuss CRNs.

2.4 Chemical Reaction Networks

Portions of this section are borrowed from [54]. The systems we use to model chemistry are known as Chemical Reaction Networks (CRNs), which is an instance of an Artificial Chemistry (AC) [55]. CRNs give us a mean to model a set of chemical species and the way in which they react with each other to form new

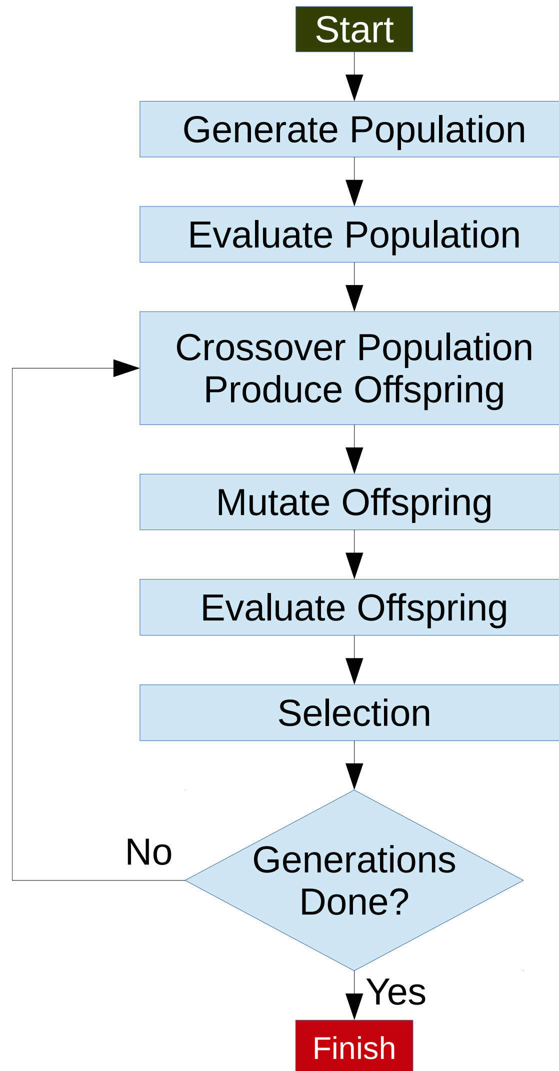


Figure 2.6: A simple genetic algorithm proposed by Bäck [43]. This algorithm builds a population and then evaluates it. Next, it uses crossover to produce a pool of offspring and then mutates and evaluates the offspring. The best individuals are selected and this process is repeated until the specified number of generations are met.

products. We model the chemistries by using several observations of nature, such as chemical kinetics and the law of conservation of mass.

A CRN consists of a set of species and reactions with associated rates. In our system, molecular species are symbolic and unstructured, and there is no notion of space because we assume the solution is well stirred. In other words, the concentration of a given species is uniform throughout the solution and not unequally distributed. Actually, we do not need to handle the position of individual molecules, but rather transform all molecules of the same type (species) using rates generated by kinetic laws: mass-action [56] [57] for regular and Michaelis-Menten [58] [59] [60] for catalytic reactions.

Dittrich [55] describes an Artificial Chemistry (AC) made up of a finite set of molecular species and a finite set of reactions. The set of molecular species are represented by symbols. For example, the symbols representing the two reactants and products in our chemical example here are S_1, S_2, and P, respectively. The reactions are formed through multiple sets of species (reaction left side) that react to form products (reaction right side) [7]. A reaction looks like $S_1 + S_2 \rightarrow P$ where reactants S_1 and S_2 form the product P.

We combine mass action kinetics with the ideas of AC to express reaction rates for ordinary (non-catalytic) reactions. Epstein [61] expresses this through a series of differential equations. Given a generic chemical reaction $aS_1 + bS_2 \rightarrow cP$, the rate of reaction, v, is expressed by

$$v = -\frac{1}{a}\frac{d[S_1]}{dt} = -\frac{1}{b}\frac{d[S_2]}{dt} = \frac{1}{c}\frac{d[P]}{dt} = k[S_1]^a[S_2]^b, \tag{2.2}$$

where $[S_1]$, $[S_2]$, and $[P]$ are the concentrations of the reactants, S_1 and S_2, and the product, P. Symbols a and b are stoichiometric constants, and k is the reaction rate

constant. Reactions could also be reversible, but in this paper, for simplification, we assume the reverse rate is always zero.

Michaelis-Menten kinetics describes the rate of a catalytic reaction where a substrate (S) is transformed into a product (P) through the use of an enzyme or catalyst (E) in a reaction modeled as $E + S \rightleftharpoons ES \rightarrow E + P$. The catalyst E speeds up the rate of the reaction without being consumed in the process. The rate of a catalytic reaction is defined by

$$v = \frac{k_{cat}[E][S]}{K_m + [S]}, \tag{2.3}$$

where k_{cat} and K_m are rate constants [62].

The simulations of CRNs were performed with COllective cELlular computing (COEL). COllective cELlular computing (COEL) is a tool developed by Banda *et al.* that allows the simulation and evolution of CRNs using GAs [63]. The chemical simulations, as we will show later, take a longer period of time to run than a non-CRN simulation so distribution of the work across a High-Performance Computing (HPC) cluster allows the simulations to execute faster. We have confidence in the results produced with multiple papers being published using the same tool. Building blocks, like perceptrons and ANNs as chemistries are also already modeled in COEL.

Various models of perceptrons that are constructed in a CRNs already exist. Banda *et al.* have presented multiple models of the preceptron such as the Asymmetric Signal Perceptron (ASP) [64] and Analog Asymmetric Signal Perceptron (AASP) [13]. These two models of perceptrons both have a definition that allows direct mapping to a biological implementation. The AASP is an improved version of the ASP that offers greater precision with a fewer number of reactions. Blount

et al. has shown that more than one of these pereceptrons, forming an ANN, in the same chemistry are possible with the use of chemical compartments [14].

These chemical compartments are a means to isolate the reaction of a series of reactions from another set of reactions. From a modularity perspective, they also allow some sense of recursion by re-using the same species and reactions set by inserting multiple copies of the same compartment. Blount's compartments use a membrane to control the flow of interactions between each of the layers of an ANN. This gives way to allow multiple perceptrons to co-exist in the same solution without interfering with the processing of another perceptron. Now, in the next chapter, we will discuss the implementation of two different models of the delay line as a CRN.

Chapter 3

Delay Line

With the knowledge of Chemical Reaction Networks (CRNs) and Genetic Algorithm (GA), we now move on to discuss the two models of delay line implemented as a CRN. First, a model of the delay line that has greater precision, but requires more control signals. Then, the second model requires fewer control signals, but it comes at the cost of precision. Both models are then connected with a Asymmetric Signal Perceptron (ASP) to demonstrate functionality and modularity. This chapter is based off our accepted paper [54].

3.1 Delay Line Concept

A delay line is a way to store data in an ordered fashion over time. The delay line we design operates similar to a First-In, First-Out (FIFO) data buffer, but allows random access to any element of the FIFO. Figure 3.1 shows an example of a delay line shifting values down. Table 3.1 shows an example of how values shift down with increasing time, t.

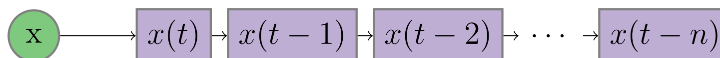


Figure 3.1: Diagram of an example delay line. The circle, x, is the input value and box $x(t)$ represents the value of x at current time step. Box labeled $x(t-1)$ represents one time step ago, $x(t-2)$ represents two time steps ago, and so on.

t	x	x(t)	x(t − 1)	x(t − 2)	x(t − 3)
0	15	15			
1	19	19	15		
2	12	12	19	15	
3	14	14	12	19	15
4	11	11	14	12	19

Table 3.1: Table of values for a given input value, x, and how they shift through a delay line. Each value, $t - n$, represents the value n time steps ago.

3.2 Delay Line Design

To introduce the time delay line design, we will first examine a delay line constructed of only two stages in two different styles. One is a Manual copy Delay Line (MDL) that requires experimenter participation to indicate when it is time to move values between stages. The second model automatically propagates the signaling species backwards, hence it is more autonomous, but it comes at the cost of additional and cumulative error in the resulting output values.

3.2.1 Manual Copy Delay Line

First, we will introduce the delay line of two stages with manual copy of the signaling species shown in Figure 3.2. A delay line of two stages is composed of seven species: X, $X1C$, $X1$, $X2$, $X2C$, $X2_{signal}$, and $X1_{signal}$. The species X represents the input value of the delay line. The signaling species, $X1_{signal}$ and $X2_{signal}$, are the catalysts that start the reaction conversion of X into corresponding stages. The primary function of $X1_{signal}$ is to trigger and accelerate the copy reaction which converts of X to $X1C$ and $X1$. Species $X2_{signal}$ performs a similar action for the conversion of $X1C$ to $X2$.

Species $X1C$ and $X2C$ are delayed copies of X that move to the next stage of the system (for example, $X1$ to $X2$ and $X2C$). Species $X2C$ is shown for

completeness and is used to cascade the system to a delay line of more than two stages. For a two stage delay line, $X2C$ is waste and flushed. The outputs of the system are the $X1$ and $X2$ species, i.e., $X1$ and $X2$ represent the current and previous values of X that are consumed as the inputs of another system.

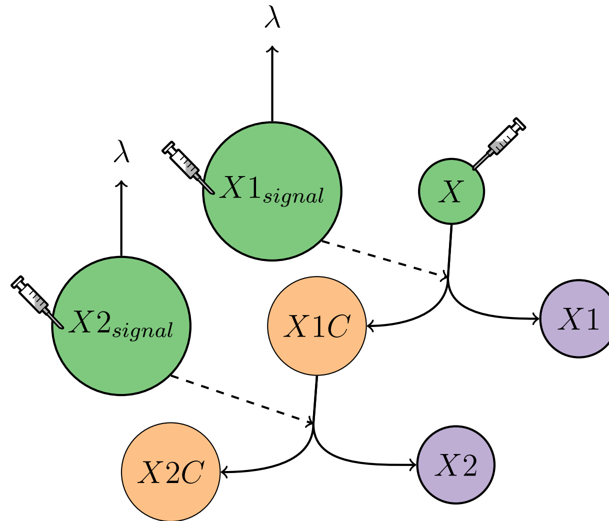


Figure 3.2: Manual copy Delay Line (MDL) with two stages. The syringe is used to indicate the species where inputs are presented and $X1$ and $X2$ represent the output species from the delay line. Species $X2C$ is used to cascade a value to a delay line of greater than two stages. The signal species, $X1_{signal}$ and $X2_{signal}$, catalyze the copy reactions and are removed from the system by decay (λ).

Species $X1C$ is the internal transition storage species. The storage species acts as a buffer for the value that will transition into $X2$ on next activation of the system with an $X2_{signal}$ passed in. Ideally, the concentration of $X1C$ will be the same as $X1$ prior to its consumption. This process is represented by a set of reactions using the previously mentioned species. Reactions 3.1 and 3.2 (below) represent the conversion of the input species, X, through to the output species,

$X1$ and $X2$.

$$X \xrightarrow{X1_{signal}} X1 + X1C \tag{3.1}$$

$$X1C \xrightarrow{X2_{signal}} X2 + X2C \tag{3.2}$$

Reactions 3.3 and 3.4 show the decay (represented by lambda, λ) of the catalyst species, $X1_{signal}$ and $X2_{signal}$.

$$X2_{signal} \rightarrow \lambda \tag{3.3}$$

$$X1_{signal} \rightarrow \lambda \tag{3.4}$$

Now, using these reactions, we can examine data moving through the delay line. For this MDL, actions must occur at two moments (in time). First, at time zero, we present a random value to the input X and reset $X1$ and $X2$ to zero. The reset of $X1$ and $X2$ simulate consumption by the underlying system the delay line is integrated with. Species $X2_{signal}$ is set to one to copy the value stored in $X1C$ to $X2$. In the ideal case for the initialization and first run of the delay line, $X2$ should be zero until these actions repeat. After 25 time steps, $X1_{signal}$ is injected to the system. The wait is to fully allow the transition of $X1C$ to $X2$ before beginning the transformation of X to $X1C$. These injections repeat every 1,000 time steps and are summarized in Table 3.2. Table 3.3 shows an example of these injections repeating every 1,000 time steps with example data moving through.

Figures 3.3 and 3.4 shows the results of running the actions in Table 3.2 for 10 iterations (10,000 time steps). Valid data is available for examination on output species $X1$ and $X2$ every time steps after each cycle. Figure 3.3a shows the input values injected to the manual delay line. During the first cycle, species $X2$ remains

Table 3.2: Actions for two stage MDL simulations.

Time	Species	Value
0	X	$0.0 \leq rand() \leq 1.0$
0	$X1$	0
0	$X2$	0
0	$X2_{Signal}$	1
25	$X1_{Signal}$	1

Table 3.3: Pipeline view of data moving through manual signaling delay line from Table 3.2. Bold items show those injected to the system. **A**, **B**, and **C** are inputs and **1** is a concentration (presence) of Xm_{signal}.

Species	Time=0	25	1000	1025	2000	2025
X	**A**	$A \rightarrow 0$	**B**	$B \rightarrow 0$	**C**	$C \rightarrow 0$
X1signal		$\mathbf{1} \rightarrow 0$		$\mathbf{1} \rightarrow 0$		$\mathbf{1} \rightarrow 0$
X2signal	$\mathbf{1} \rightarrow 0$		$\mathbf{1} \rightarrow 0$		$\mathbf{1} \rightarrow 0$	
X1	**0**	$0 \rightarrow A$	**0**	$0 \rightarrow B$	**0**	$0 \rightarrow C$
X1C		$\rightarrow A$	$A \rightarrow 0$	$0 \rightarrow B$	$B \rightarrow 0$	$0 \rightarrow C$
X2	**0**		$\mathbf{0} \rightarrow A$	A	$\mathbf{0} \rightarrow B$	B
X2C			$\rightarrow A$	A	$A \rightarrow B$	B

at zero since there is no previous value as seen in Figure 3.3b. Figure 3.4a shows the catalysts, $X2_{signal}$ and $X1_{signal}$, sequentially injected each cycle. Figure 3.4b presents the sequence of actions where $X2_{signal}$ is injected at time zero followed by $X1_{signal}$ 25 time steps later.

3.2.2 Backwards Signal Propagation Delay Line

The Backwards signal Propagation delay Line (BPL) handles the signal species differently. More specifically, the only input signaling species is $X2_{signal}$ and rather than decay, $X2_{signal}$ reacts to $X1_{signal}$. The advantage of this model is that the user is only required to perform actions at the beginning of the cycle and then the system transforms the species internally (without external help). Figure 3.5 shows a revision of the MDL for this model. This reduces the number of injections to

two: the input (X) and the final copy signal ($X2_{signal}$ for two stage). The change leaves reactions 3.5 and 3.6 unchanged.

$$X \xrightarrow{X1_{signal}} X1 + X1C \tag{3.5}$$

$$X1C \xrightarrow{X2_{signal}} X2 + X2C \tag{3.6}$$

Revising the remaining reactions requires modifying only reaction 3.3. Removing the decay from reaction 3.3 so that $X2_{signal}$ reacts to $X1_{signal}$ gives the updated reactions 3.7 and 3.8.

$$X2_{signal} \rightarrow X1_{signal} \tag{3.7}$$

$$X1_{signal} \rightarrow \lambda \tag{3.8}$$

All actions in the system occur instantaneously and are the same as actions employed by the manual delay line at time zero. At the beginning of every cycle, $X1$ and $X2$ are set to zero to simulate the next block of the system consuming their values. Also, a random value (X) and signal ($X2_{signal}$) are injected to the system. Table 3.4 summarizes these actions, which repeat every 1,000 time steps to ensure enough time for all reactions to reach steady state.

The simulations of the Backwards signal Propagation delay Line (BPL) run for 10,000 time steps (same as for the manual delay line). Valid data is also produced at the same point (every 50 steps) on the output species $X1$ and $X2$. The value produced on the first cycle of $X2$ ideally should be zero, but leakage from $X1C$ is generally seen from steps zero to 1,000 (see Figure 3.6b). An input is introduced

Table 3.4: Actions for two stage back propagation delay line simulations. These actions repeat every 1000 similar to Table 3.3.

Time	Species	Value
0	X	$0.0 \leq rand() \leq 1.0$
0	$X1$	0
0	$X2$	0
0	$X2_{Signal}$	1

to the system at species X (Figure 3.6a) and then is reacted in the same cycle to species $X1$ (Figure 3.6b). After the next cycle (i.e., the next introduction of $X2_{signal}$), the value injected at X previously is now presented at $X2$ (Figure 3.6b).

Notice that the backwards propagation introduces an error to the system with some of the $X2$ values not lining up exactly with the previous $X1$. This difference is due to the time window that the reactions for X to $X1$ and $X1$ to $X2$ are simultaneously active. Looking at Figure 3.7b, $X1_{signal}$ and $X2_{signal}$ are large enough for both catalyses to occur. So, for this small window of time, there is effectively a direct path from X to cascade down to $X2$. This overlap is not inherently a problem. It allows the desired parallelism of this system. We can afford this error in a small number of stages, but the inaccuracy can grow with a larger number of stages.

3.2.3 Inherit Single Instruction, Multiple Data

With the nature of chemistry, one of the advantages of our unconventional delay line implementation is the ability to perform single instruction, multiple data [65] operations. The main factor is finding a unique set of species to hold each delay line that will not react with surrounding buffers to allow such parallel operations. Figure 3.8 shows an example of a two-stage set of backwards propagation and MDLs that are producing a vector of three values for the current and previous

cycles.

3.2.4 More than Two Stages

Extending the buffer for more than two stages is straightforward. For each stage we add one output species (Xm), transition species (XmC), and catalyst species (Xm_{signal}). This allows the system to flexibly provide a buffer of desired length. As an example, Figure 3.9 shows a BPL with three stages. The total number of species required in the system grows at a rate of $3m + 1$, where m is equal to the number of stages in the system. One trade-off to note is that as the number of stages in the system increases, so does the period of time to cascade values through the delay line. Ideally, each reaction runs to full completion prior to Xm_{signal} propagating backwards to begin the next conversion.

The reaction set of the delay line also scales in a straightforward fashion. Each intermediate delay stage has a reaction similar to reaction 3.5 and the final delay stage (the mth delay) has a reaction similar to reaction 3.6. This remains true for extending both the manual and backwards propagating delay line. Extension of the catalysts depends on the implementation. For the MDL, simply adding the species and a subsequent input is required. Extending the backward propagating delay line has the advantage that it does not increase the number of injections, but it still increases the overall number of species.

3.3 Results

We will highlight the results for the two stage buffer and its extension beyond two stages. We employed GAs [39] to optimize the rate constants (mapped to chromosomes) of the backwards propagation model. We only used the algorithm to optimize the backwards propagation model since the manual copy was straightforward to optimize by hand. The GA used an elite selection of the top 20 chromosomes from the population of 100, which undergo cross-over and mutation to form the next generation. The goal (fitness function) of this evolutionary algorithm was to minimize the error of the delay line.

We defined error as the difference between the actual input value (X) and the value occurring at $X1$ on this cycle and then $X2$ on the next cycle. We performed this test 50 time steps after X is injected into the delay line. Equation 3.9 shows the calculation of this error where $X[n]$ represents the current value of X and $X[n-1]$ represents the value of X on the previous input cycle. Adding both differences for the two stage delay line provided the overall error.

$$error = |X_1 - X[n]| + |X_2 - X[n-1]| \tag{3.9}$$

The genetic algorithm performed perturbation mutation that changed each chromosome's element with 30% chance by ±30% using a uniform distribution. We ran the GA for 100 generations to produce the results for the two stage delay line. The algorithm was configured to target a transition of the input species, X, to the current time species, $X1$, as fast as possible, and convert the intermediate species, $X1C$ to the previous time species, $X2$, as fast as possible while minimizing the amount of leakage between the phases of the design.

3.3.1 Two Stages

Table 3.5 shows the rate constants for the manual propagation delay line reactions. Rates for the conversion of input species, X, down the chain is the same rate with the presence of $X1_{signal}$ and $X2_{signal}$ both increasing the rate by the same amount because the forward copy reactions should be as fast as possible. Figures 3.3 and 3.4 shows the plots using these rate constants in a two stage system, which can be replicated for a manual copy system of any size.

Table 3.5: Rate constants of two stage MDL found by GA.

Reaction	Forward Rate	K_m
$X \xrightarrow{X1_{signal}} X1 + X1C$	0.0757	2.0000
$X1C \xrightarrow{X2_{signal}} X2 + X2C$	0.0757	2.0000
$X2_{signal} \to \lambda$	0.5643	(None)
$X1_{signal} \to \lambda$	0.5643	(None)

For a different size, the back propagation delay line has different rate constants. In addition, the rate constants were not grouped like the manual propagation delay line because it would drastically decrease the performance. Looking at the constants in Table 3.6, the reaction for species $X1C$ to $X2$ is the fastest. This is directly due to the rapid rate that $X2_{signal}$ is reacting to $X1_{signal}$. Effectively, to meet the first requirement of getting X into $X1$ as fast as possible, the lower level transition of species (Reaction 3.6) must complete before. Figures 3.6 and 3.7 shows the output of a two stage BPL with the rate constants in Table 3.6.

To compare the accumulated error of the two delay lines we used Symmetric Mean Absolute Percentage Error (SMAPE) defined as

$$SMAPE = 100 * \langle \frac{|y - \hat{y}|}{y + \hat{y}} \rangle, \tag{3.10}$$

Table 3.6: Rate constants of two stage BPL found by GA.

Reaction	Forward Rate	K_m
$X \xrightarrow{X1_{signal}} X1 + X1C$	0.0020	0.0225
$X1C \xrightarrow{X2_{signal}} X2 + X2C$	0.0706	2.0000
$X2_{signal} \rightarrow X1_{signal}$	1.3648	(None)
$X1_{signal} \rightarrow \lambda$	0.0039	(None)

where $\langle . \rangle$ is the mean of a set of multiple values, y is the actual value, and $\hat{y}$ is the expected value. We calculated an average SMAPE per stage (unit size) by dividing cumulative SMAPE with m. More specifically, using n to represent a discrete time series sample and m to represent the number of stages:

$$SMAPE = \frac{100}{m} * \sum_{k=1}^{m} \langle \frac{|Xk - X[n-(k-1)]|}{Xk + X[n-(k-1)]} \rangle. \tag{3.11}$$

For instance, SMAPE for two stages ($m = 2$) is given by

$$SMAPE = \frac{100}{2} * \langle \frac{|X1 - X[n]|}{X1 + X[n]} + \frac{|X2 - X[n-1]|}{X2 + X[n-1]} \rangle. \tag{3.12}$$

We evaluated performance (error) over 10,000 runs, each repeating the sequence of actions defined in Table 3.2 and Table 3.4 for 200 iterations (200,000 time steps). Figure 3.10 shows the results for a delay line of size two as well as for larger sizes (discussed in next section). The difference in values from expected values for the two stage delay line are quite small. This shows that for a two stage delay line, both types operate well. One thing to note is that the backwards delay line has a larger initial error which can accumulate over time.

3.3.2 Over Two Stages

In this section, we will examine the use of a delay line with five stages. Five stages was selected and executed for both the manual copy and back propagating delay line. Figure 3.10 shows the final error when evaluated for 10,000 runs for 200 iterations each (same as for $m = 2$ in previous section). The maximum error over the entire evaluation is shown in Table 3.7. There are a few observations to note on this plot. The error on a backwards propagation delay line (B) increases as the number of stages in the delay line increases. For a smaller delay line, this error would generally be negligible, but for larger sizes this could be a concern. The manual copy has a significantly smaller error as shown in Figure 3.10.

Table 3.7: Maximum and average SMAPE obtained through performance runs of 200 iterations and varying configurations of stages and manual copy and backwards propagation. Maximum and average excludes the initial values where the delay line is filling (first m points with low SAMP).

Backwards DL	Max	Average	Manual DL	Max	Average
B5	14.35%	14.09%	M10	0.0059%	0.0016%
B4	11.66%	11.25%	M5	0.0049%	0.0024%
B3	5.26%	4.84%	M2	0.0033%	0.0008%
B2	2.28%	1.97%			

As for the backwards propagating delay line, the error starts to accumulate to a noticeable value rapidly. Even by phase three, the delay line is starting to produce error that is in excess of the MDL with ten stages. Looking back to Figure 3.7b there is a period of time where both $X1_{signal}$ and $X2_{signal}$ overlap which can explain how error that starts quite small in stage one of the delay system accumulates to a large value by the time it reaches the later stages of the buffer. Depending on the desired properties of the delay line, this is worth considering for the application.

3.3.3 Time Series Perceptron Integration

To demonstrate the capabilities of the delay line to fit into other designs, we integrated it with a chemical perceptron called a threshold asymmetric signal perceptron introduced by Banda [64]. This perceptron learns through reinforcements and is inspired by biological neurons. Integration with the delay line and the perceptron shows how the delay line can easily fit with other systems to act as an input stream without any design modifications. Previously, the perceptron received both values simultaneously as two inputs. Now, we are showing that, without change to the perceptron or delay line, the two integrate together and function well. Figure 3.11 shows an example of this integration.

We trained the perceptron using reinforcement learning on 14 linearly separable binary functions. Figure 3.12 shows the results of this learning. The binary time series perceptron learns 11 of the 14 functions with an accuracy of greater than 85%. Figure 3.13 shows the buffer and perceptron accurately producing the output for OR.

NAND, IMPL, and NOTX1 are all heavily dependent on the last input to resolve in the time delay line, $X1$. In this case, the input species $X1$ is not provided to the system until typically 50 time steps later than value $X2$. The original model of the perceptron was optimized for instantaneous and simultaneous injection of both inputs. Because input $X1$ is not ready, the performance is lower because that input plays a larger role on the correct performance for these logic functions. This makes the system capable of obtaining an average success rate of approximately 90% compared to the perceptron's 99% success rate [64].

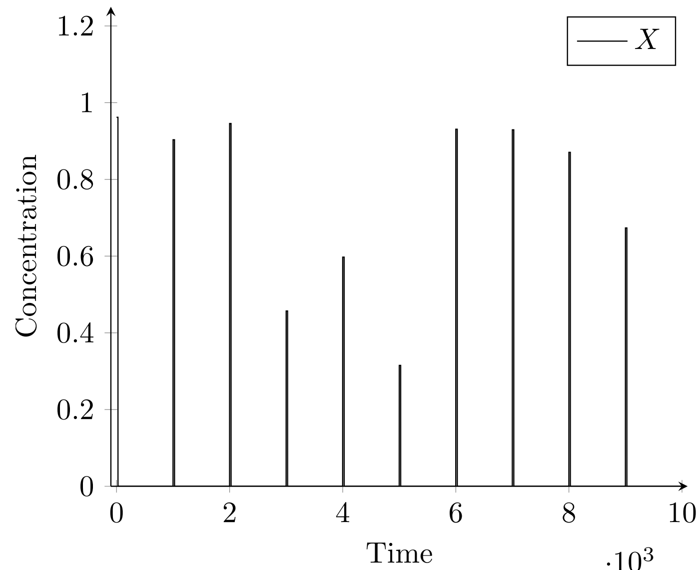
3.4 Discussion

We have presented a novel implementation of a delay line as a chemical reaction network capable of storing past concentrations. By arranging our delay lines in a SIMD-like layout, we could delay multiple segments of data simultaneously with a shared control signal for either model of delay line. We have introduced two types of a chemical delay line: manual copy delay line and backwards propagation delay line. A manual copy delay line can precisely carry values in a delayed state, but requires more intervention from the user (growing at a rate of m) to propagate values through the system. The second model, backwards propagating delay line, automatically moves values through the system with a single signaling injection with reasonable accuracy.

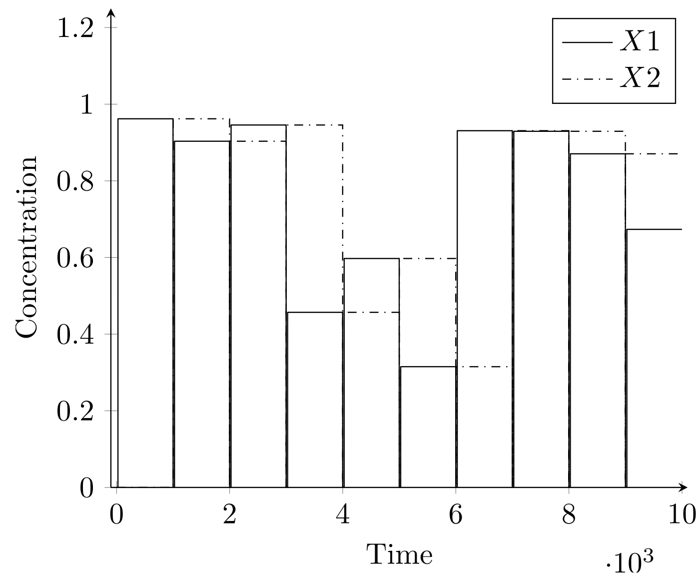
The integration of the backwards propagating delay line with the threshold asymmetric signal perceptron resulted in the first chemical model capable of learning binary time series. Also, this example is a proof-of-concept that our delay line is a modular block ready for use in other systems. For systems requiring a smaller window of past values, either model of the delay line gives sufficient accuracy for data storage. The manual copy delay line shows potential for longer chains with the amount of calculated SMAPE passed between phases remaining below 0.01% for a delay line of 10 stages. The backwards propagating delay line provides a much simpler user interface at the sacrifice of accuracy. A backwards propagation of five stages keeps the calculated SMAPE below 15%. Systems requiring a large number of delays will have to weigh accuracy and simplicity to make a selection for a particular implementation.

The BPL and MDL tied with a ASP demonstrate how these two systems can modularly connect to other elements in a CRN to form a memory. This connection

of our delay line model shows how the delay line can modularly connect to an independently developed component, like an ASP, to solve larger problems. Using Blount's compartments [14] as an XOR Artificial Neural Network (ANN) combined with a model of our delay line would allow for the development of a block like a Linear Feedback Shift Register (LFSR). Now that we have the building blocks to provide data storage in a CRN, we discuss the work to find the optimal size of memory and layout of ANN to solve the ant trail task when paired with a delay line.

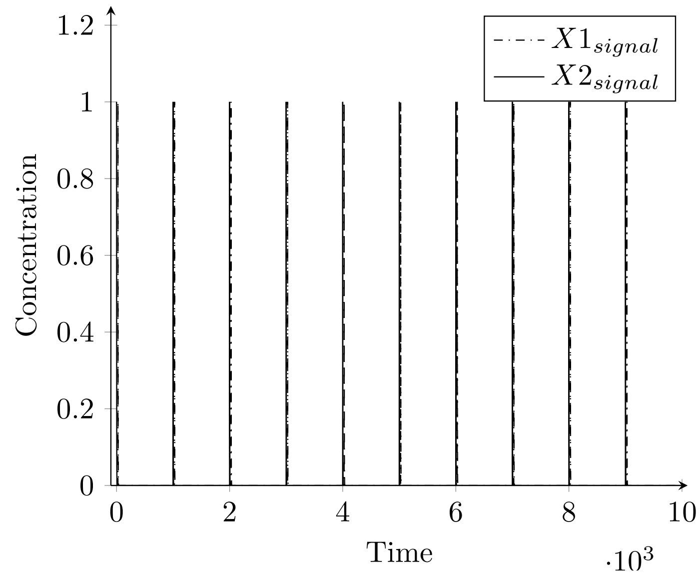


(a) Input

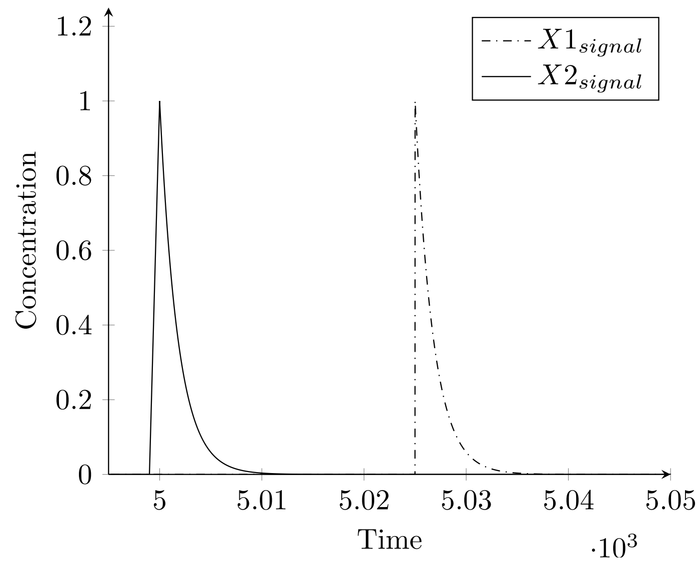


(b) Outputs

Figure 3.3: Two stage MDL showing input and output signals. Data arrives as input (3.3a) and is available on outputs (3.3b) with $X1$ being the current and $X2$ being the previous X.



(a) Copy Signals



(b) Copy Signals (Zoomed on Figure 3.4a)

Figure 3.4: Two stage MDL showing the copy signals. The copy of this data is triggered by $X1_{signal}$ and $X2_{signal}$ (3.4a). Figure 3.4b shows the copy signals zoomed in from Figure 3.4a.

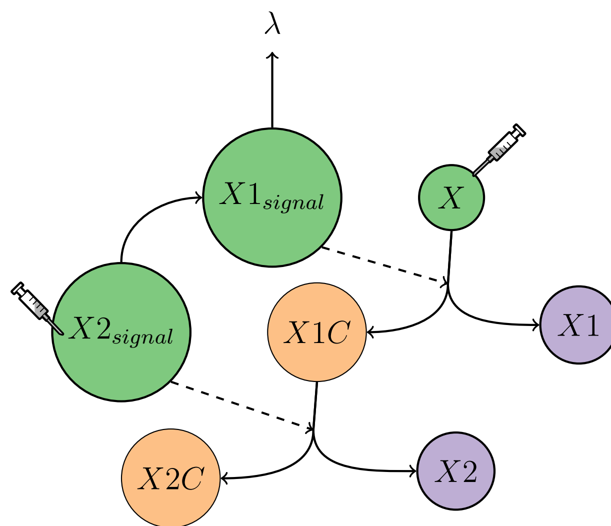
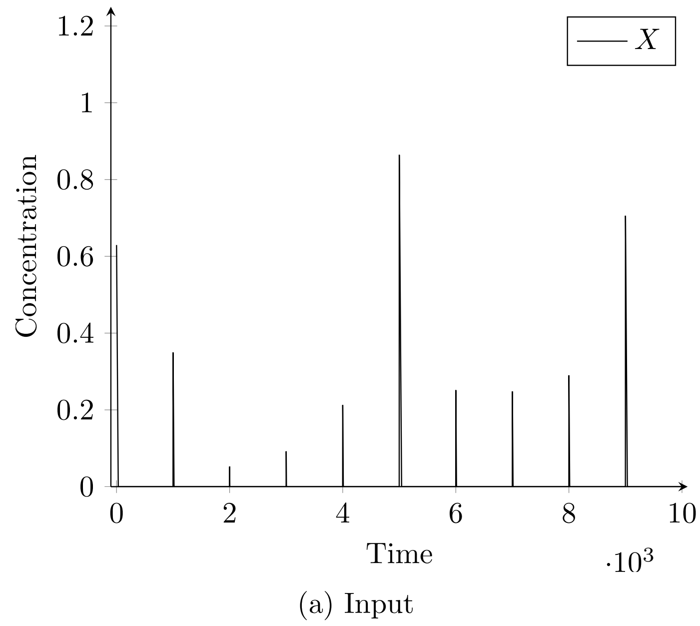
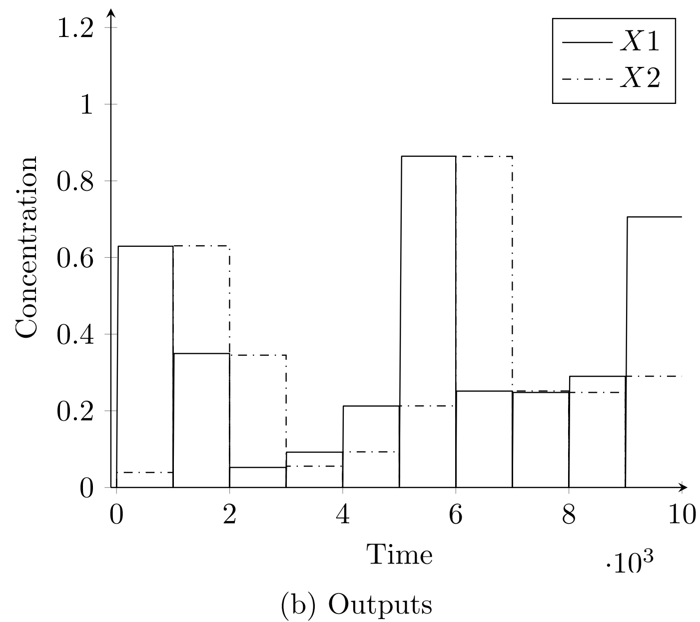


Figure 3.5: Backwards propagating delay design with two stages. The syringe is used to indicate an injection of the input species X and the copy signal $X2_{signal}$. The species $X1$ and $X1$ represent the output species from the delay line. The signal $X2_{signal}$ is propagated backwards to $X1_{signal}$ without user intervention and then decays (λ).

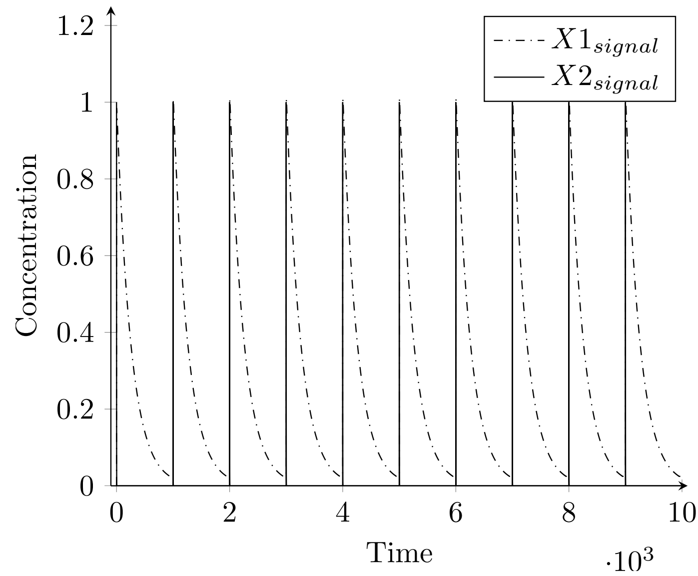


(a) Input

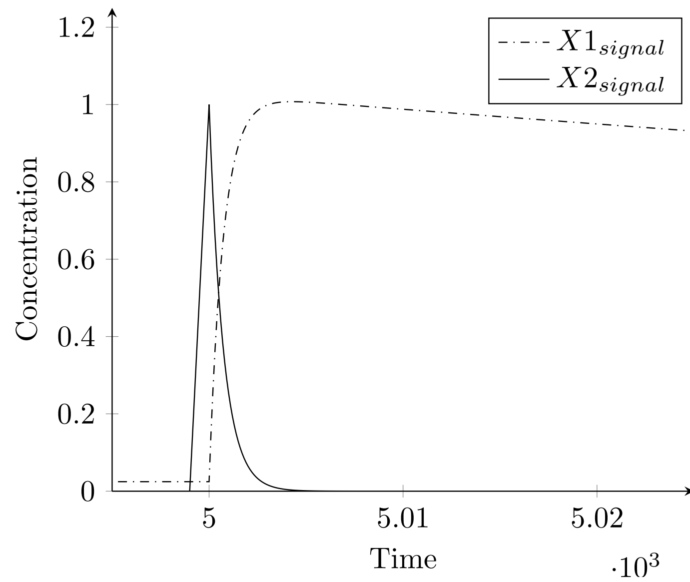


(b) Outputs

Figure 3.6: Two stage backwards propagation delay line showing inputs and outputs. Data arrives as input (3.6a) and is available on outputs (3.6b) with $X1$ being the current and $X2$ being the previous X.



(a) Copy Signals



(b) Copy Signals (Zoomed on Figure 3.7a)

Figure 3.7: Two stage backwards propagation delay line showing copy signaling. The copy is started by $X1_{signal}$ and $X2_{signal}$ (3.7a). Figure 3.7b shows the signals controlling propagation zoomed in from Figure 3.7a.

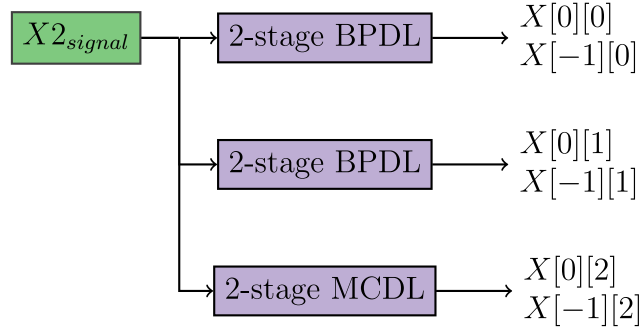


Figure 3.8: Time delay design Single Instruction, Multiple Data (SIMD) representation showing simultaneous output of previous ($X[-1][n]$) and current ($X[0][n]$) X for parallel data processing. The signaling can be used with multiple instances of a delay line, both for the manual copy and the back propagation type.

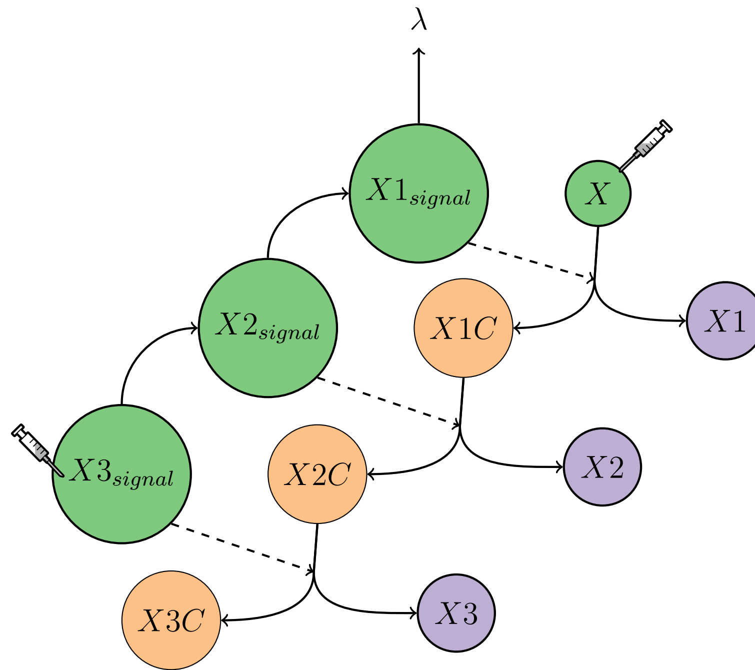


Figure 3.9: BPL with three stages. The syringe is used to indicate an injection of the input X and the signal $X3_{signal}$. Species $X1$, $X2$, and $X3$ represent the output species from the delay line. Lambda (λ) shows decay of backwards propagation signal.

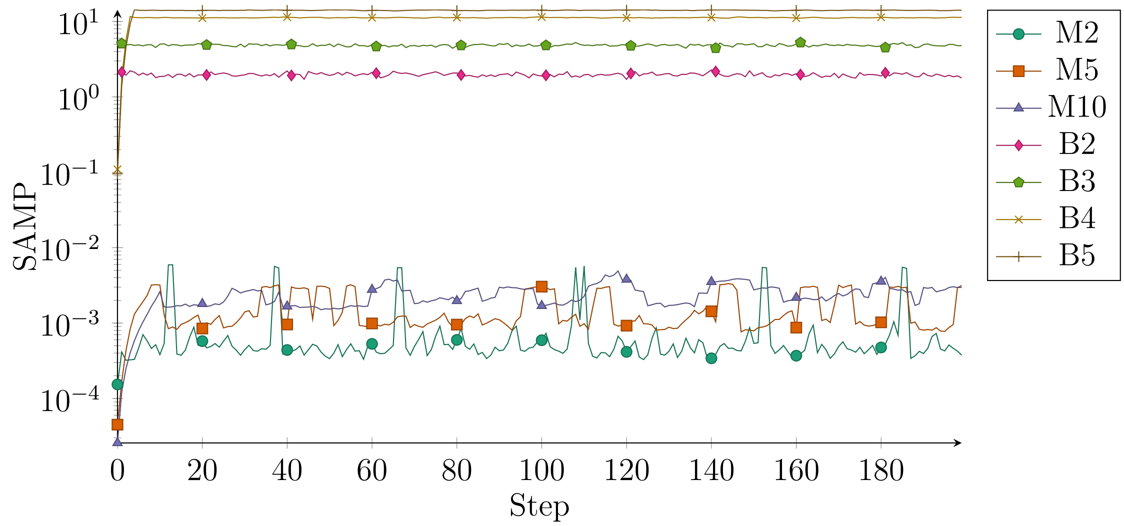


Figure 3.10: SMAPE calculated for delay lines. Mm and Bm are the mth stage of manual copying and back propagation delay line.

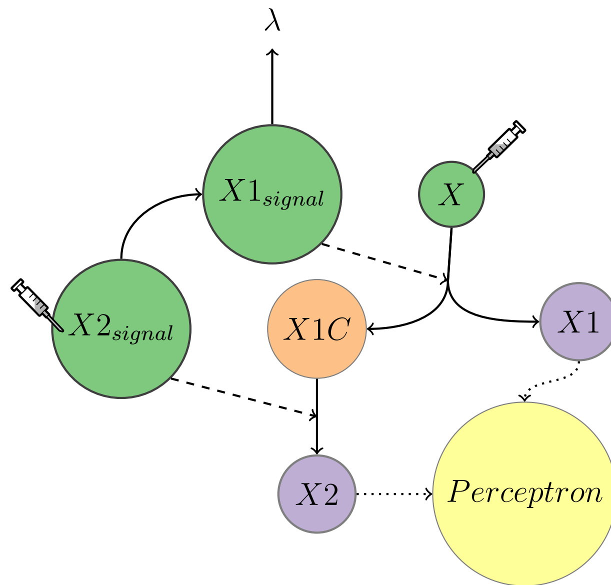


Figure 3.11: Perceptron integration with backwards propagating delay line of two stages. The delay line outputs ($X1$ and $X2$) are fed to the perceptron without modification of the delay line.

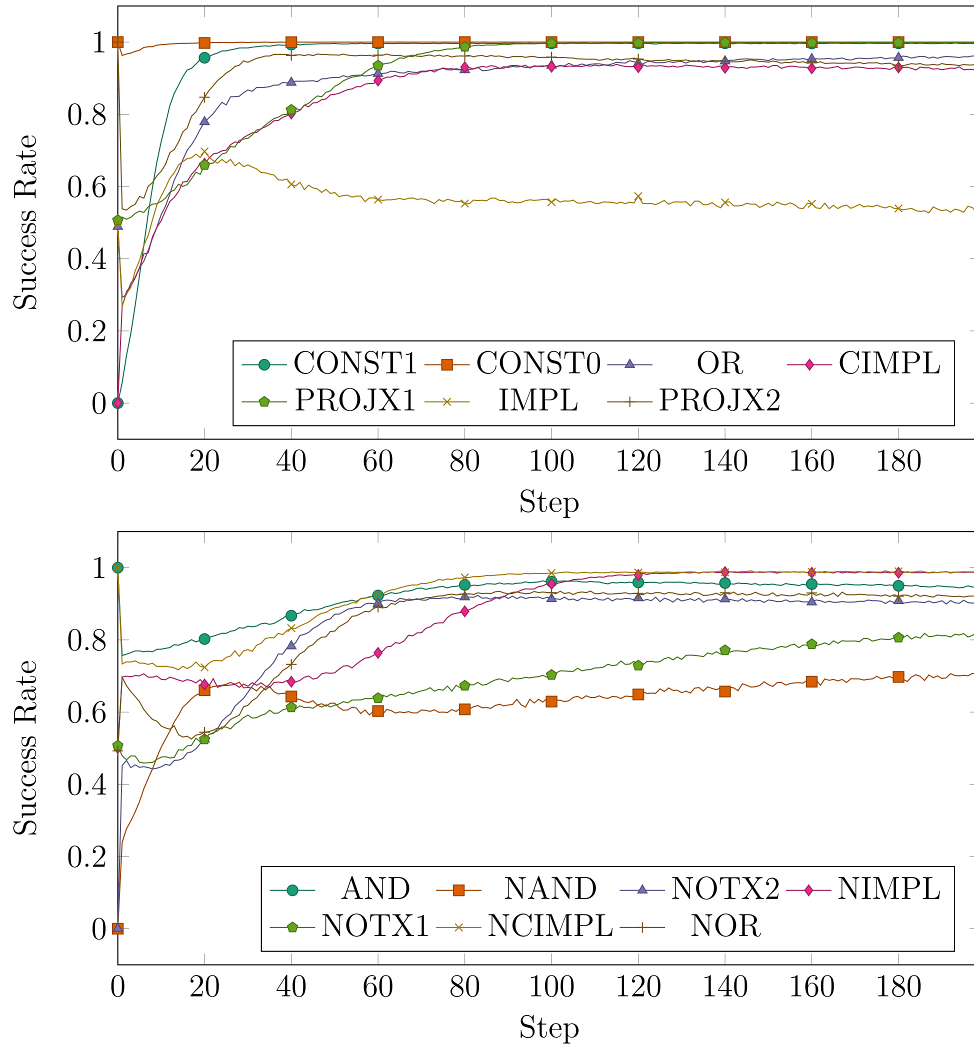
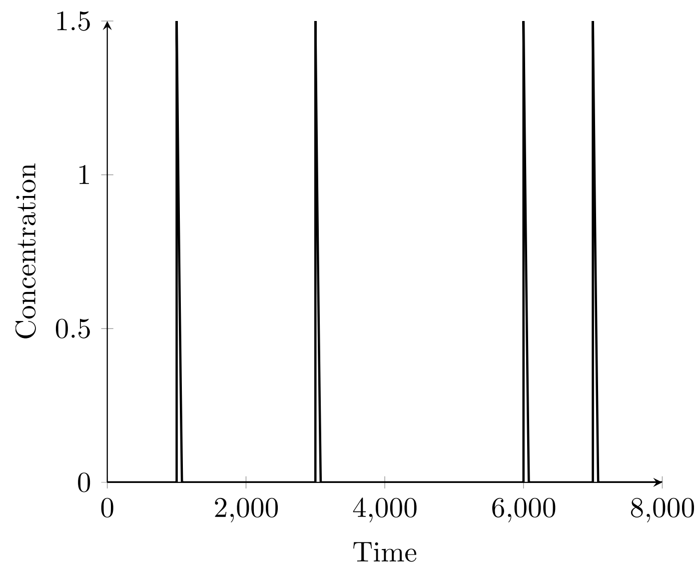
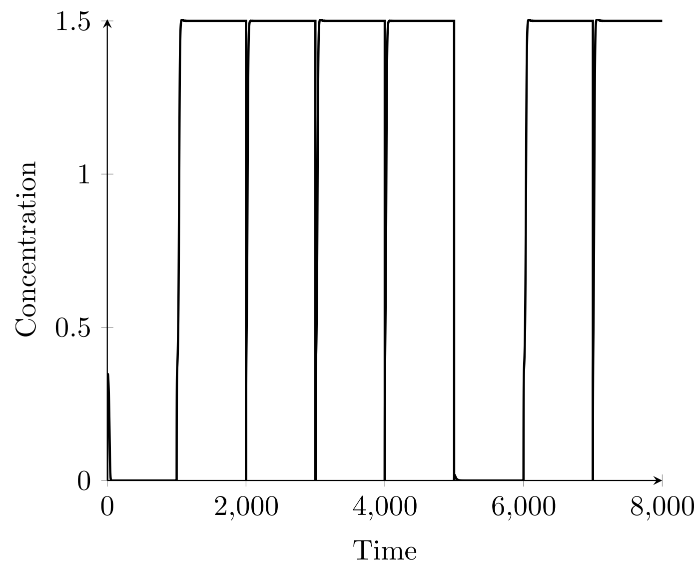


Figure 3.12: Success rate of binary time series chemical perceptron. The perceptron learns 11 of the 14 linearly separable functions with an accuracy of greater than 85%.



(a) Input Stream



(b) Output Stream

Figure 3.13: Example concentration traces of binary time series chemical perceptron that successfully learns OR function. Left shows input stream 0,1,0,1,0,0,1,1. Right shows correct output stream of 0,1,1,1,1,0,1,1. Two zeros on the input stream at 4,000 and 5,000 successfully produce zero at time 5,000 on output stream.

Chapter 4

Trail Runner and Trail Viewer

This chapter discusses the pair of tools developed for simulation and analysis of the John Muir, Santa Fe, and other trails prior to moving to a Chemical Reaction Network (CRN) environment. We found no other application that readily performed the evaluation required for our work so we developed two applications to aid in our research. The two applications, trail runner and trail viewer, are built on published, open-source tools and were designed with extensibility in mind. Trail runner is a parallel Genetic Algorithm (GA) trail evaluator that manipulates parameters of the simulations and records the runs to a database. Trail viewer is a web-based application that allows users to filter, browse, and view the results from various types of trail simulations. This framework allowed us to easily sweep across different parameters to locate an ideal configuration for later implementation in a CRN. In this chapter, we will discuss both tools, some of the advantages of these tools, and their framework.

4.1 Trail Runner

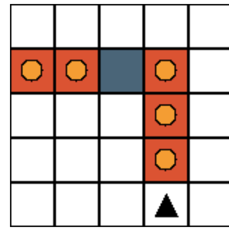
Trail runner is responsible for evaluation of different networks to see performance against navigating the agent through different trails (like the John Muir, Santa Fe, and more). Trail runner also performed evolution of the parameters on the networks with GAs working towards the maximum performance possible. We have also published trail runner under an open source license and it is available for download at `https://github.com/jmoles/trail-runner`. The application is

a command line-based Python tool that uses published tools for evaluation of the Artificial Neural Networks (ANNs) through evolutions in a GA.

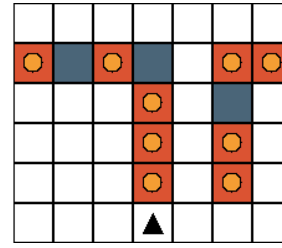
The ANNs in trail runner were modeled with PyBrain, a machine learning library for Python [66]. PyBrain served as the basic model for the different flavors of ANNs. The evolution of the parameters on the ANNs is performed with Distributed Evolutionary Algorithms in Python (DEAP) [67]. DEAP is a powerful toolbox allowing straightforward evaluation of GA in Python. One of the primary reasons for the selection of DEAP was the tightly integrated use of a distributed computing framework, Scalable COncurrent Operations in Python (SCOOP) [68]. SCOOP allowed the distributing of work across multiple servers in the lab cluster for reduced run time. With the use of PyBrain, DEAP, and SCOOP, we then had to build a series of tools to evaluate the different trails and report the results.

The intent of trail runner is to allow evaluation of different GA and ANN parameters to attempt and find an ideal solution prior to implementation in a CRN. Evaluating the same ANN in a CRN (with a tool like COllective cELlular computing (COEL) [63]) can take a substantially longer amount of time. Performing the same optimization in a CRN from the beginning would have caused the amount of time to simulate this project to increase dramatically. Later, we will show some examples of the speedup with SCOOP and running the same network in COEL. Easily sweeping across different parameters in trail runner allowed us to narrow the set of potential parameters to run simulations against.

Running the trail runner application is relatively straightforward after users have configured a database for the application to record results and retrieve available configuration parameters. Trail runner uses PostgreSQL [69] and includes a script to create the initial database and populate it with the networks and trails



(a) Test trail 1



(b) Test trail 2

Figure 4.1: The two trails used for initial testing of trail runner. They are designed for fast evaluation as well as a small complication (turns, gaps) so that some GA optimization is necessary. Test trail 1 is simpler than the test trail 2 on the right.

used in this work. A safe set of defaults are specified for many of the GA parameters, but they are all customizable by the user using command line flags. A few of the key parameters as an example include the trail ID, population size, maximum moves, network ID, and generations to run for. A full list of the available parameters are available in the application's help (`--help`).

Trail runner took advantage of these tools for evaluation of the trail task. Testing on the tools was performed by first starting with a smaller trail shown in Figure 4.1. Afterwards, we moved on by testing the larger John Muir trail and got results similar to that of Jefferson's. Section 5.2 discusses the results of testing against test trail 1, test trail 2, and the John Muir trail.

4.2 Trail Viewer

Trail viewer is a Flask [70] web-based application that allows viewing of results from trail runs over their GA evaluation. Diagrams of results are rendered with the help of matplotlib [71] and Plotly [72]. The trail diagrams (like the ones in Figure 4.1) used throughout this work are also rendered using trail viewer. At the time of this writing, an instance of trail viewer is operating at `http://`

`codeboxide.joshmoles.com`. Figure 4.2 shows a screen shot of the trail viewer application. Trail viewer is also available under an open source license for download at `https://github.com/jmoles/trail-viewer`.

Early versions of trail viewer were a desktop-based Graphical User Interface (GUI). The application was migrated to a web-based tool for greater compatibility with different operating systems as well as allowing interaction with other web-based tools (like COEL [63] and Plotly [72]). Trail viewer also interacts with the same database that trail runner writes results from each run. Trail viewer only consumes information from the database and does not perform any modification. Animation of the agent navigating through the trail is also available in trail viewer. Trail viewer uses JavaScript to render the agent navigating through the trail with a specified trail and moves. The diagrams showing the moves the agent traveled (for example, Figure 5.5) were captured with the help of this capability.

The pair of these applications provide an environment to evaluate different types of networks and trail problems with numerous sets of parameters and view the results. The integration of PyBrain, DEAP, and SCOOP create a platform that allows others to easily run research similar to that of Jefferson or Koza on the John Muir, Santa Fe, or other trail of choosing. Users can easily add their own databases, networks, or GA parameters and generate plots showing a single configuration run, an average of all runs with these parameters, or evaluate a sweep of different parameters, such as delay line length.

In the next chapter, we present the use of trail runner and trail viewer to arrive at the minimal delay line length for implementation in a CRN.

Simulation Run 7226

Information

Date	Thu Oct 9 11:45:06 2014
Time	00:04:59
Host	rhone.ece.pdx.edu
Debug	False
Run Configuration	433

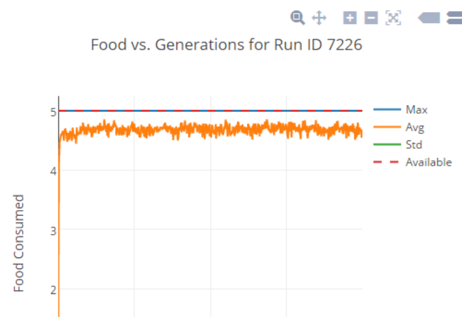
Network Name	Jefferson NN (2,5,4) v1
Trail Name	Sample L Left 5x5
Mutate Type	mutFlipBit
Generations	800
Population	200
Variations Type	varAnd
Algorithm Version	1

Moves Limit	10
Selection Method	Tournament
Tournament Size	20
P(Mutate)	0.3
P(Crossover)	0.5
Weight	[-5.0, 5.0]

Charts

"Std" indicates the standard deviation.

Food vs. Generations



Moves vs. Generations

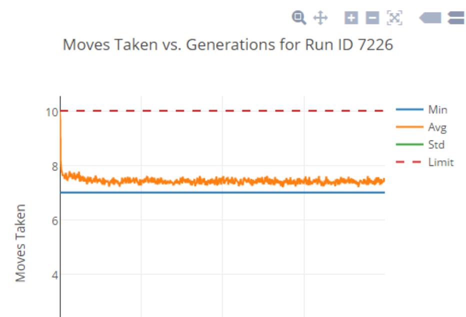


Figure 4.2: Screen shot of the trail viewer application showing the results of a single simulation run. The information of the trail and simulation configuration is shown at the top with the partial charts of results shown below.

Chapter 5

Non-Chemical Reaction Network Simulations

This chapter looks at the testing performed using trail runner and trail viewer in a non-Chemical Reaction Network (CRN). We will go over a set of test cases that were first preformed to validate that trail runner and trail viewer were presenting accurate results. The next set of tests will move toward identifying the minimal length of delay line necessary to evaluate the trail for later implementation as a CRN. We first cover the methodology we used for the testing and present the results. The chapter concludes with a discussion of the results.

5.1 Methodology

In this section, we will outline how we verified that trail runner functioned as expected and later used to find the optimal length of delay line. We did the evaluation testing on three trails: test trail 1, test trail 2 (Figure 4.1), and the John Muir Trail (Figure 2.1). We developed a genetic algorithm based off a simple straightforward algorithm proposed by Bäck [43]. A version of this algorithm was already implemented in Distributed Evolutionary Algorithms in Python (DEAP), known as `varAnd` [67]; however, the algorithm did not have all of our desired functionality. Namely, we wanted the ability to monitor the progress of the Genetic Algorithm (GA) in real time, so we used it as a basis for our GA. It also did not directly support the selection method we used, $(\mu + \lambda)$ selection [47]. Figure 5.1 shows a flowchart of this process discussed that you may find helpful while reading this section.

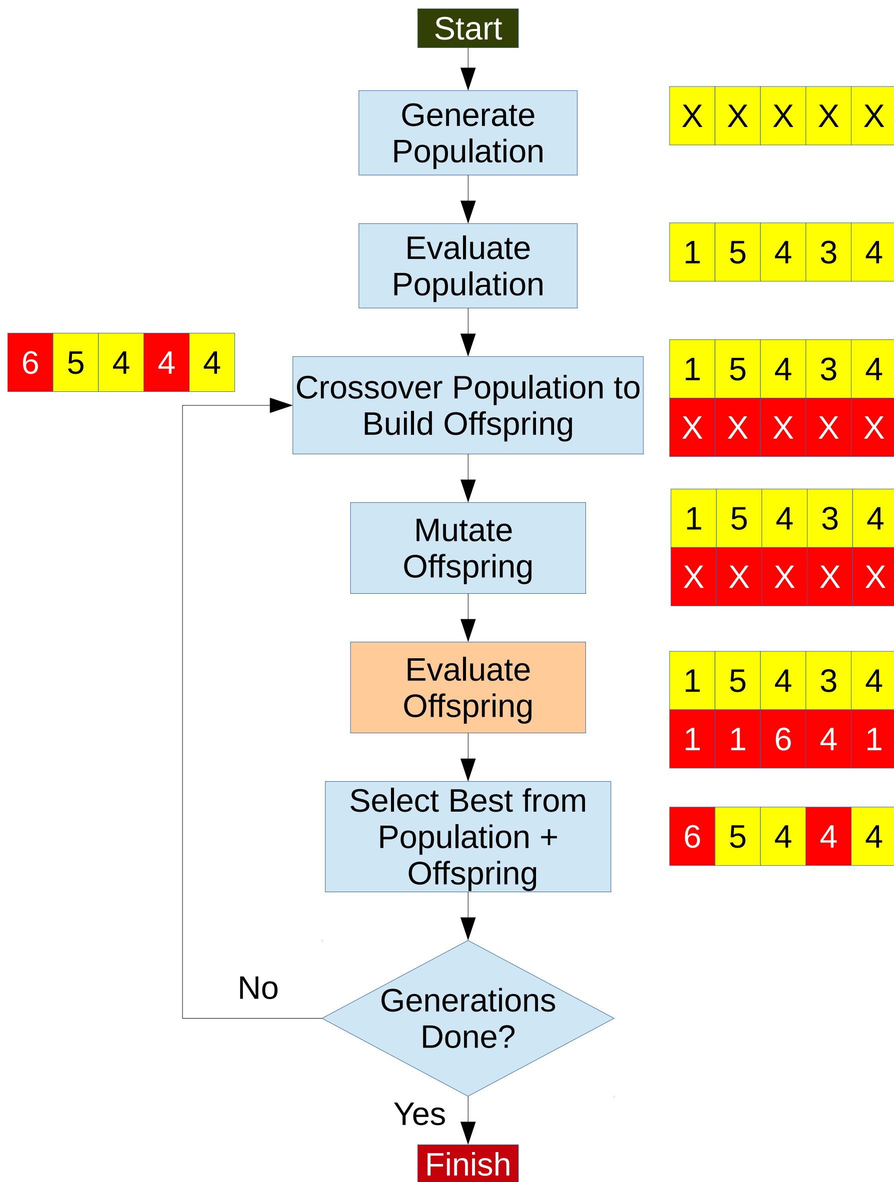


Figure 5.1: Flowchart of the GA used to test the trails. Based off the algorithm proposed by [43] and [67]. The boxes to the right and left each represent an individual through each phase of the GA. An X indicates the fitness is unknown and a number represents a fitness (higher is better). The pool to the left of crossover population represents the individuals selected and looping back on the "No" path from "Generations Done".

The GA we used started by generating a population with a size, P, defined by the user. The individuals in this pool are a set of real-valued vectors that are in the period between the minimum weight and maximum weight, or $[w_{min}, w_{max}]$. Each of these values corresponded to a weight connecting perceptrons on the Artificial Neural Network (ANN). Each individual is then evaluated to determine their fitness by running them for M moves through the specified trail. The intent was to find all pieces of food before focusing on minimizing the number of foods. We wanted to place greater emphasis on the amount of food consumed, so fitness (f) is calculated by:

$$f = 1.0 \times food\,consumed - 0.1 \times moves\,taken \tag{5.1}$$

Afterwards, we crossover the population with a two-point crossover [49] with a specified probability of undergoing crossover, p_x. If an individual does not get selected for crossover, it is copied to the pool of offspring.

Next, a Gaussian mutation method is used on the pool of offspring because of it's common use for real-valued vectors [43]. All individuals enter mutation in this algorithm and each real value of the chromosome is adjusted with a probability of mutation, p_m. Then, these offspring are evaluated and assigned their fitnesses before entering selection. Our selection selected the best P individuals from the pool of offspring and the original population. This selection method is also known as $(\mu + \lambda)$ [47]. We then continue to repeat this process until one of the three criteria for exiting are met:

- the user specified number of generations (G) are evaluated,
- all food in the trail is consumed, or

- or there has been no change in the standard deviation of the mean fitness (m_g) of the individual pool for the previous g generations.

The final termination criteria was added to save processing time for this task. We found that if the average was stuck at this point for a long period of time, the likelihood of one of these runs proceeding to increase fitness was low. Also, the amount of time in general to solve this problem was rather large so terminating an evaluation that is not progressing is more advantageous than waiting. Note that by terminating when all food is consumed means no optimization for the number of moves taken. The goal was to consume all of the food in the provided number of moves. We will show an example in the results where we terminated a run early with no change in mean fitness.

The algorithm itself had several parameters, such as population, probabilities, and generations, that we must specify. Values were selected based of recent publications as a starting point for evaluation of the GA. We selected a population, P, of 100 with a probability of crossover, p_x, of 0.6 based off work by De Jong *et al.* [73]. For the test trails, we selected a smaller population of 10 to reduce the probability of a solution at generation 1. In other words, we wanted to force the GA to operate rather than potentially randomly finding a good candidate on the first run for such a small trail. We selected a probability of mutation, p_m, of 0.05 from De Jong's thesis work [49]. Weights in the range of $[-5.0, 5.0]$ were selected for the ANNs. We set the number of generations (G) to a relatively high value of 5000 generations because we generally found that either all food was consumed or the value of m_g settled and exited early prior to reaching 5000 generations. A summary of these parameters is shown in Table 5.1.

Using this GA configuration, we tested the GA against test trail 1, test trail

Parameter	**Value**
Population (P)	100 (10 for Test Trail)
Probability of Mutation (p_m)	0.05
Probability of Crossover (p_x)	0.6
Generations (G)	5000
Mean Fitness Generations (g)	300
Weight Range ($[w_{min}, w_{max}]$)	$[-5.0, 5.0]$

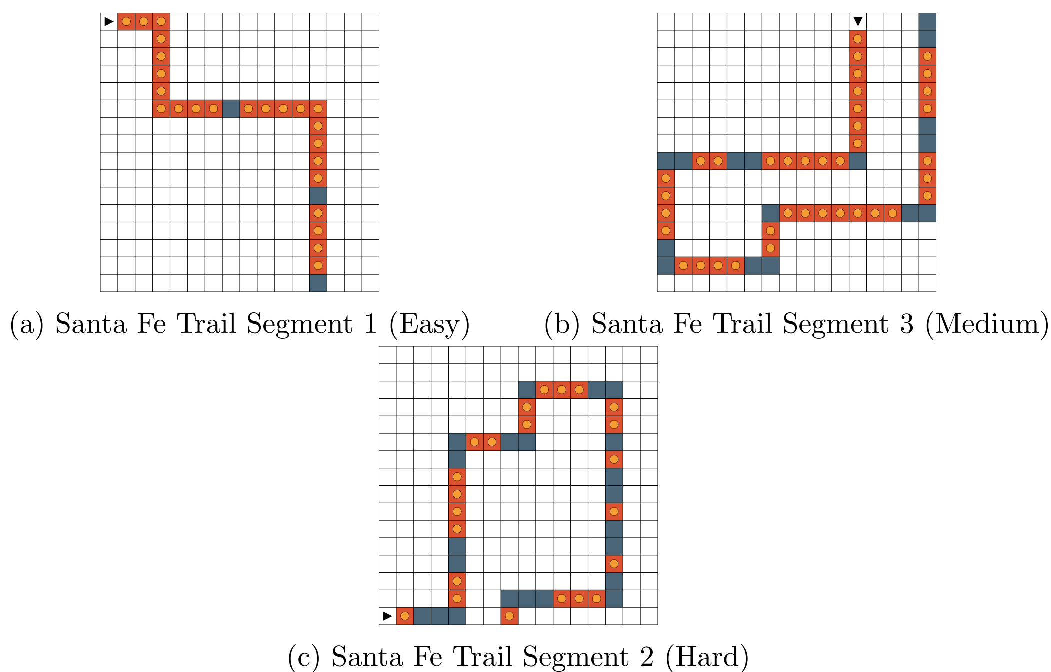
Table 5.1: This table summarizes the parameters used for the test runs in this chapter. The population and probabilities are based of work by De Jong in [73] and [49]. The generations, mean fitness generations, and weight range are set to allow ample exploration.

2, John Muir trail, and performance evaluations on the Santa Fe trail. For each trail, we limited the number of moves to 10, 20, 100, and 200, respectively. The test trail values is based off the minimum number of moves necessary plus a small overhead. The values for the John Muir and Santa Fe trail are the same used by Jefferson [12] and Koza [24]. These values are summarized in Table 5.2. For this verification exercise, we use an ANN with the same structure as Jefferson's when he solved the John Muir trail.

Trail	M	**Min. Moves**	**Food**	**Max. Fitness**
Test 1	10	7	5	4.3
Test 2	20	14	9	7.6
John Muir	200	147	89	74.3
Santa Fe	400	165	89	72.5

Table 5.2: We show the statistics for the trails used in this chapter. The move limits (M) for test trails are slightly more than the minimum number of moves for each and the John Muir and Santa Fe the same values as Jefferson and Koza. The maximum fitness is calculated with Equation 5.1.

Next, we were looking to find the minimal length of delay line connected to the smallest ANN for consuming the most amount of food in the trail. To reduce the time required for each evolution, we started evaluation on three trails that are a subset of the Santa Fe trail [24] as well as testing on the full trail. The Santa



(a) Santa Fe Trail Segment 1 (Easy) (b) Santa Fe Trail Segment 3 (Medium)

(c) Santa Fe Trail Segment 2 (Hard)

Figure 5.2: The three trails with increasing difficult that were used for the delay line length optimization. The number of gaps and turns increases with each trail is the determination for the difficulty.

Fe trail was selected as the main evaluation trail in a chemistry because Koza claimed it is a more difficult trial. It is also a more common trail in literature today for testing. Figure 5.2 show segments 1, 2, and 3 of the Santa Fe trail. They were extracted as the first portions of the full Santa Fe trail that the agent would normally navigate through.

We used the same GA configuration as mentioned above. The moves limit was divided by four ($M = 400/4 = 100$) since each trail segment represents a 16 x 16 area which is a quarter of the full Santa Fe trail. Keeping all other parameters the same, we then started going across delay line lengths of $N = 2$ all the way up to $N = 16$ to search for the minimal length of delay line.

Like Jefferson's neural network, we had to have a "food ahead" and "not food ahead" to activate the ANN in the case of no food ahead. Every delay line segment

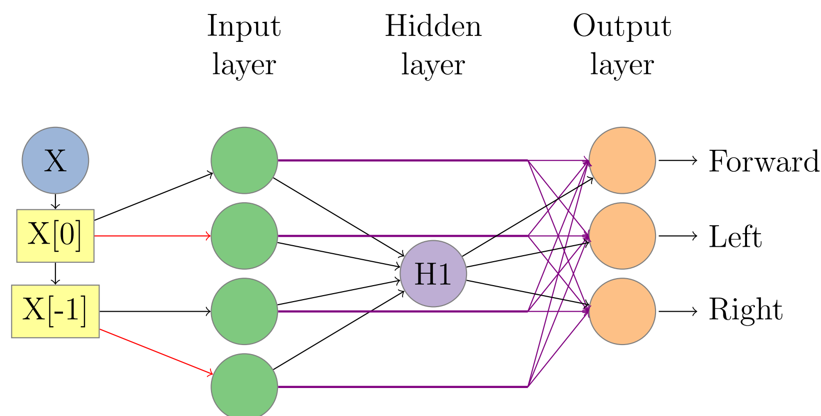


Figure 5.3: This figure shows the ANN combined with a 2-input ($N = 2$) delay line (on left). This feedforward ANN has full connections from the input to the hidden layer, input to the output layer, and hidden to the output layer. They delay line has two input neurons for each stage: one that is the actual value (black line) and one that is the inverted value (red line) of if there is food ahead. Values shift down the delay line where $X[0]$ represents the current input and $X[-1]$ represents the previous input.

required two input nodes to hold values. An example ANN with a delay line of length two is shown in Figure 5.3. There are a couple of changes from the neural network originally used by Jefferson (see Figure 2.2). Our experiments found that only one hidden node in the ANN was sufficient to solve the task when paired with a delay line. Like Jefferson and Koza, we also found that the “None” (meaning no move) output from the ANN was not used by the best individuals [12] [24]. So, the number of total nodes in the system is represented by Equation 5.2.

$$n = 2 \times delay\ line\ length + 4 \tag{5.2}$$

5.2 Results

This presents the results from running testing against test trail 1, test trail 2, John Muir trail, Santa Fe trail, and the three segments. For all of the food consumption

plots in this section, "Max" shows the food consumed by the best individual, "Min" shows the food consumed by the worst individual, and the average of the maximum of the entire population represented by "Avg". "Available" shows the maximum amount of food available in the trail.

The first set of results are from test trail 1. Test trail 1 was configured to run with the configuration specified in tables 5.1 and 5.2. We executed a small set of runs (five) with the same configuration. In many cases, an optimal GA was found after the first couple generations, but we have presented one here that has a diverse population with varying maximum, mean, and minimum. Figure 5.4 shows the food consumed versus generations and Figure 5.5 shows the path that the agent at elite individual at the final generation (eight in this case) took to consume the maximum amount of food.

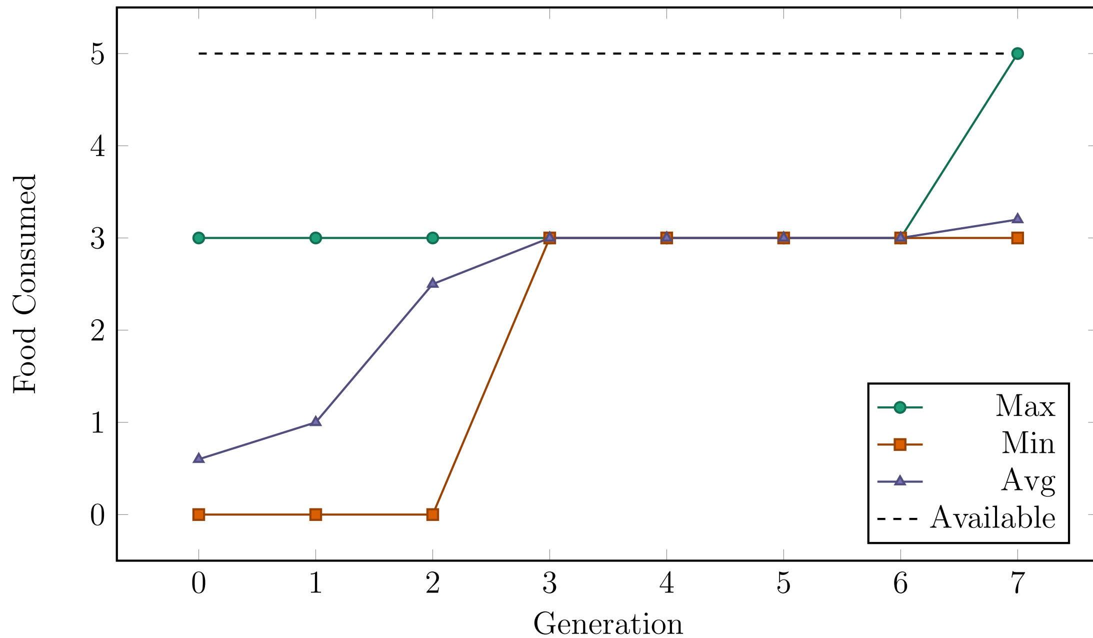


Figure 5.4: Plot showing the food consumed over each generation for test trail 1. With this relatively simple trail, it takes only eight generations to find an individual capable of consuming all the food.

Figure 5.5: This figure shows the path the best individual took with test trail 1 indicated with the dark line. Note this trail has two optimal paths (left or right at first turn). The particular solution from the GA here took the right turn. Video available at `http://goo.gl/MUktKs`.

Next, we evaluated test trail 2. Test trail 2 is a slightly more complicated version of test trail 1 featuring two turns, three gaps, and nine pieces of food. This trail is also larger than test trail 1 and the optimal path is to turn left at the first turn versus test trail 1 the agent can turn left or right. This trail was executed with the same configuration with test trail 1 except for allowing twice as many moves, twenty instead of ten. Figure 5.6 shows how much food the agent consumed and Figure 5.7 shows the path that the final individual took through the maze. Note in this case, the agent actually took extra, unnecessary steps to consume all of the food.

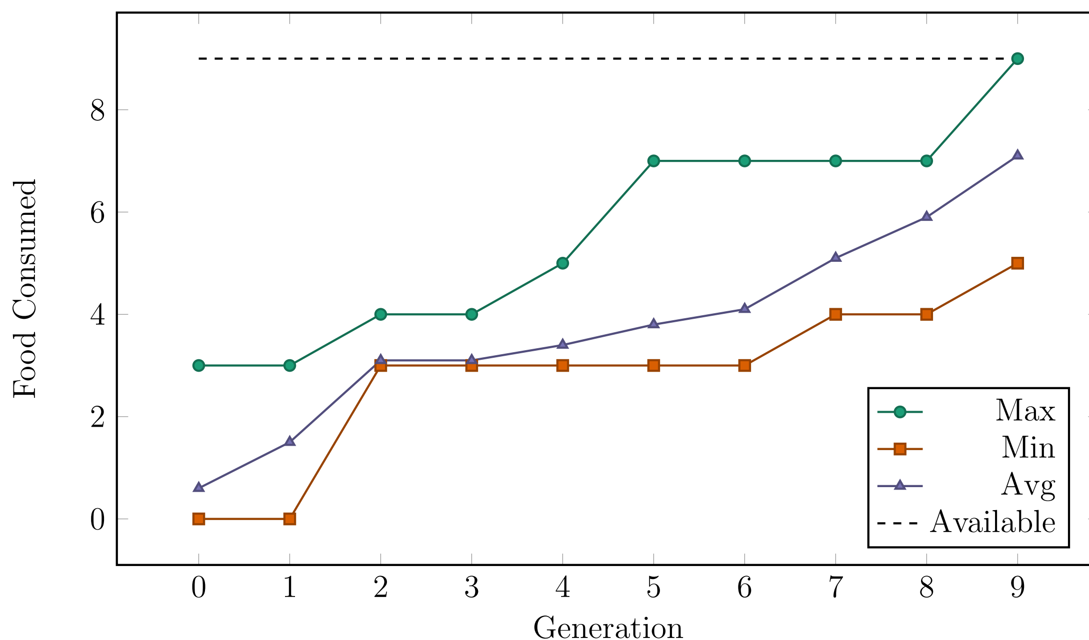


Figure 5.6: Plot of food consumed over each generation on test trail 2. This slightly more complicated trail takes a few more generations, but finds a solution after ten generations.

Figure 5.7: Path of agent consuming food in test trail 2 in final generation. In this case, the ant found a solution consuming all food, but did it with a non-optimal number of moves. As we mentioned before, we stop when a solution that finds all food is found and do not continue to optimize for moves. Video available at `http://goo.gl/jDO7p1`.

The next GA evaluation was performed on Jefferson's John Muir trail. This is a complicated trail compared to the two test trails and serves as a good starting benchmark for the performance of the GA. Figure 5.8 shows the food consumption over generations. Generation 443 contained an individual that consumed all of the food on the trail. Figure 5.9 shows the path this individual took through the John Muir trail.

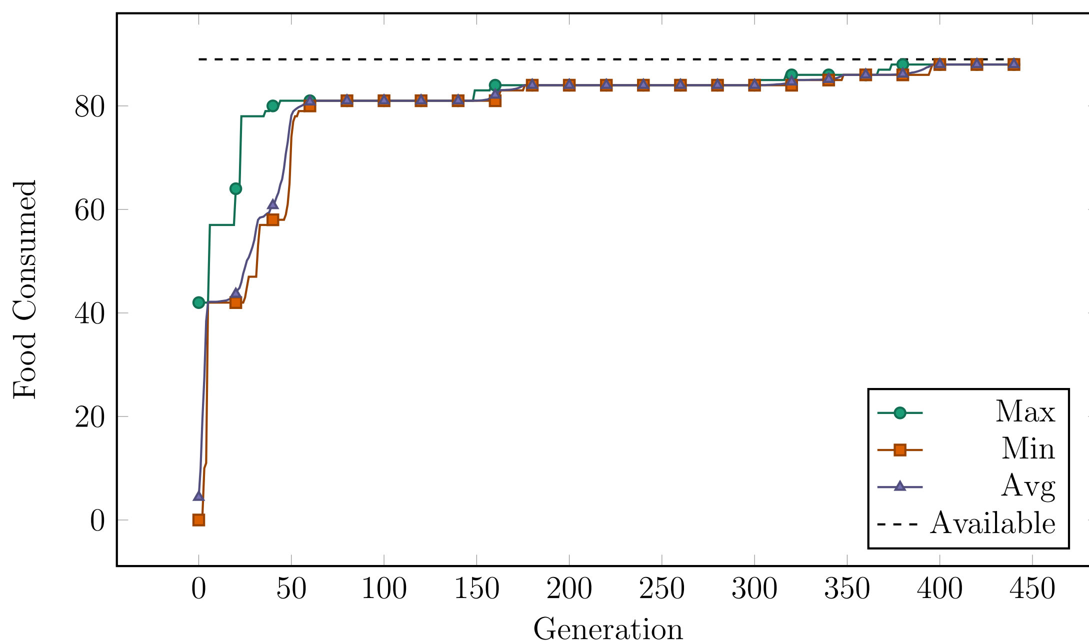


Figure 5.8: Plot showing the consumption of food for each generation on the John Muir trail. This trail took more generations compared to test trail 1 and test trail 2 to reach a solution because of the larger size. It finds a solution after 443 generations.

Figure 5.9: Path of the agent through Santa Fe Trail in final (443) generation. The agent did turn within some squares before moving forward like the first left turn on the trail. This individual required 199 moves and consumed all 89 pieces of food. The weights on this GA run evolved an interesting behavior. This individual never took a left turn and only made right turns through the trail. Video available at `http://goo.gl/OaGsyh`.

Using the same configuration on the John Muir trail, Figure 5.10 shows an example of a run that was terminated early. This run did not go to completion because there was no change in the standard deviation of the maximum amount of food consumed across the population.

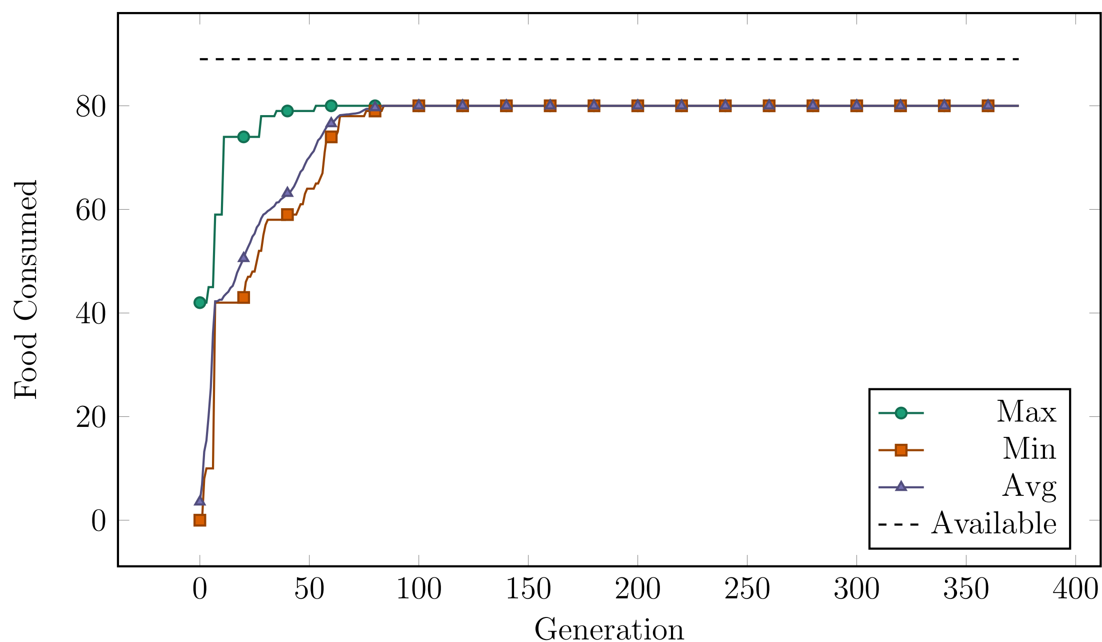


Figure 5.10: Plot showing a GA run on the John Muir trail that is no longer progressing. This run was terminated early because it did not have a change in the standard deviation of the maximum food conumsed for the previous 300 generations starting at around generation 90.

Now, we will show the results of a performance sweep to see the advantage of using a parallel processing environment, like Scalable COncurrent Operations in Python (SCOOP). We performed this evaluation using the Santa Fe Trail and and used the GA parameters specified in Table 5.1. For these trials, we disabled the automatic termination if all food was consumed for a fair comparison. All of these runs ran for the full 100 generations. Then, the number of moves was ran with a value of 100, 200, 300, and 400 across a maximum number of processes of 1, 2, 4, and 8. Figure 5.11 shows this plot. This benchmark was performed on a DigitalOcean Virtual Private Server (VPS) featuring a 160 GB Solid State Drive (SSD), 16 GB of memory, and an eight core processor [74].

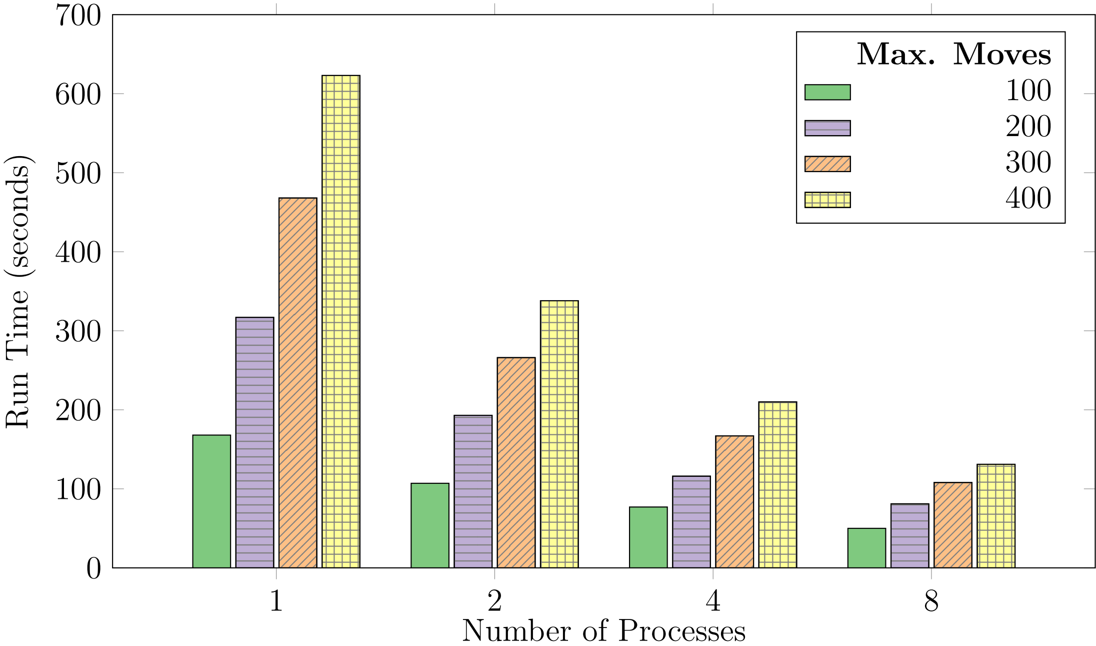


Figure 5.11: Run time benchmark for trail runner on Santa Fe trail. Benchmark swept the number of processes and maximum number of moves to collect time for each one. Notice how more processes speeds up the simulation and more moves requires more time to process.

Figure 5.12 shows a run where the “None” move option was enabled to show how it is not used by the best individual at the end of the run. This run goes for 89 generations until an individual evolves that consumes all pieces of food. This was performed on the full Santa Fe trail.

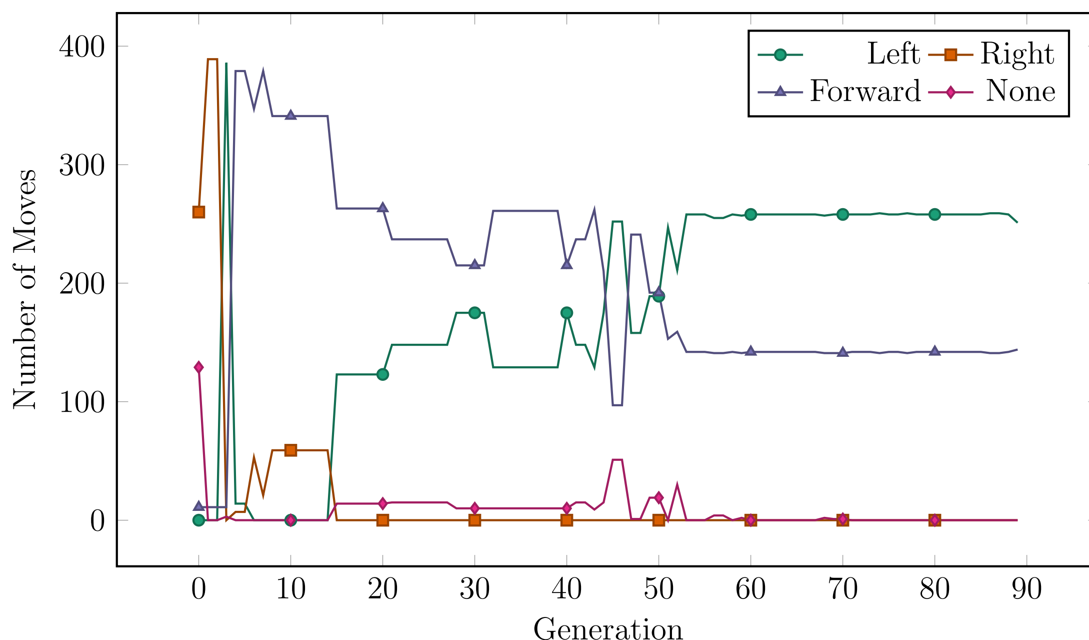


Figure 5.12: Plot showing the moves the best individual took over a run of 89 moves where it ended with an individual who consumed all food. Notice how the individuals starting around generation 70 took no “None” actions. This individual also evolved a final strategy where it only took left turns. This was a common theme in the Jefferson ANN solutions where optimal individuals would be biased to make only one type of turns, left or right.

We next swept the delay line length from $N = 2$ up to $N = 16$ on each of the three segments of the Santa Fe Trail. Figure 5.13 shows the results on the food collected by the best individual for each of the three segments and the full Santa Fe trail. This chart is showing the best individual run out of a minimum of 70 GA evaluations for every delay line length and every trail segment. There are a minimum of 25 evaluations for each length of the full Santa Fe trail. The values are normalized by dividing the food obtained by the maximum amount of food on the trail.

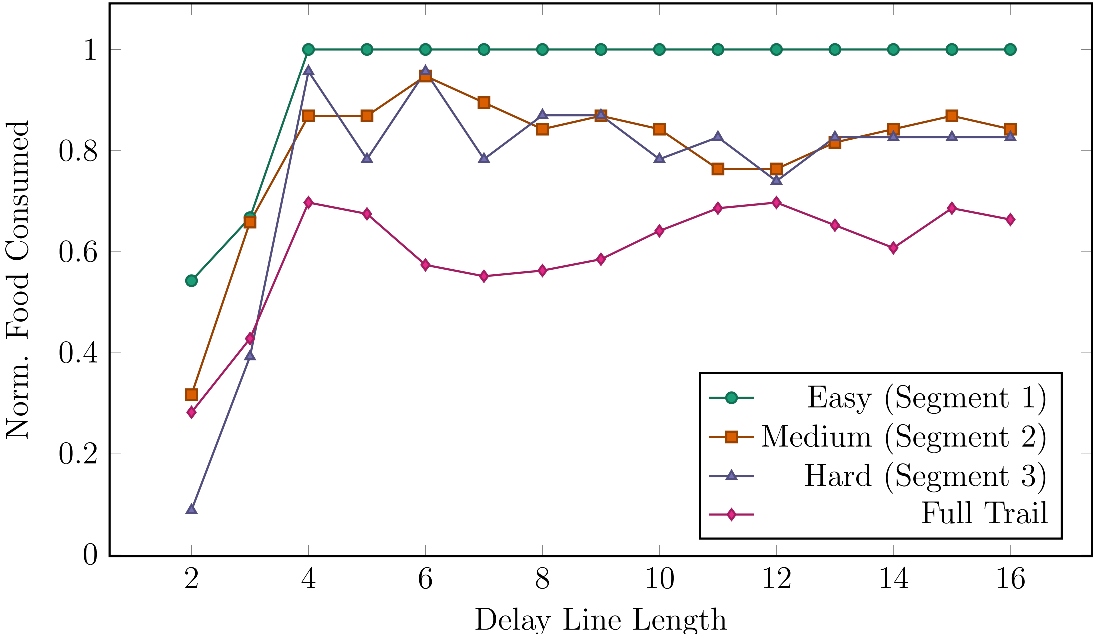


Figure 5.13: This plot shows the normalized food gathered by the best individual for a given run with varying delay line lengths. Each segment is a subset of the Santa Fe trail. Values were normalized by dividing the food gathered by maximum available for each trail. Notice how a Manual copy Delay Line (MDL) of length 2 or 3 is not sufficient. At length 4, the agent start collecting at least 60% of the available food for all segments and the full trail. The full trail is a more difficult task compared to the segments because there is a much larger trail to explore provided a larger chance for the agent to make poor moves.

Figure 5.14, Figure 5.15, Figure 5.16, and Figure 5.17 show the normalized mean for the food collected across each GA evaluation's best individual. Figure 5.18 shows all four stacked without error bars. The error bars on each chart represent the standard deviation from the means for the food collected across each GA evaluation's best individual. These charts are useful when determining the minimum delay line length for each trail.

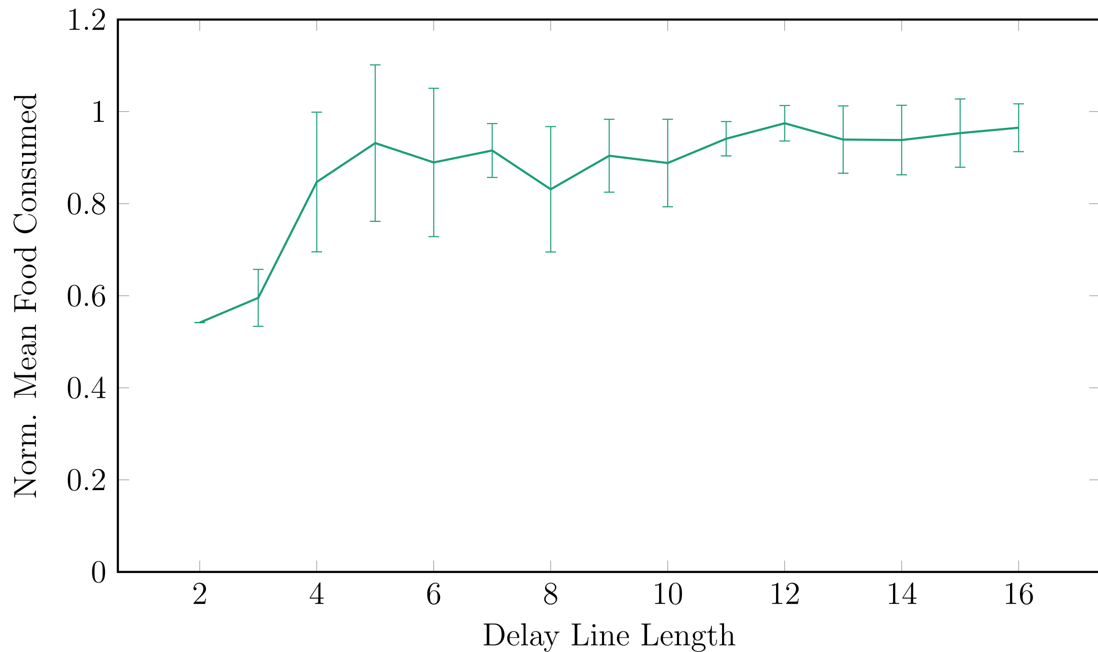


Figure 5.14: Plot showing the normalized food gathered with the average of the best individual across each GA run on the easy trail segment. Values are normalized by dividing the food gathered by the maximum amount on the trail. Local maximums are at 5 and 12, but the wide standard deviations make it hard to draw conclusions of this chart alone.

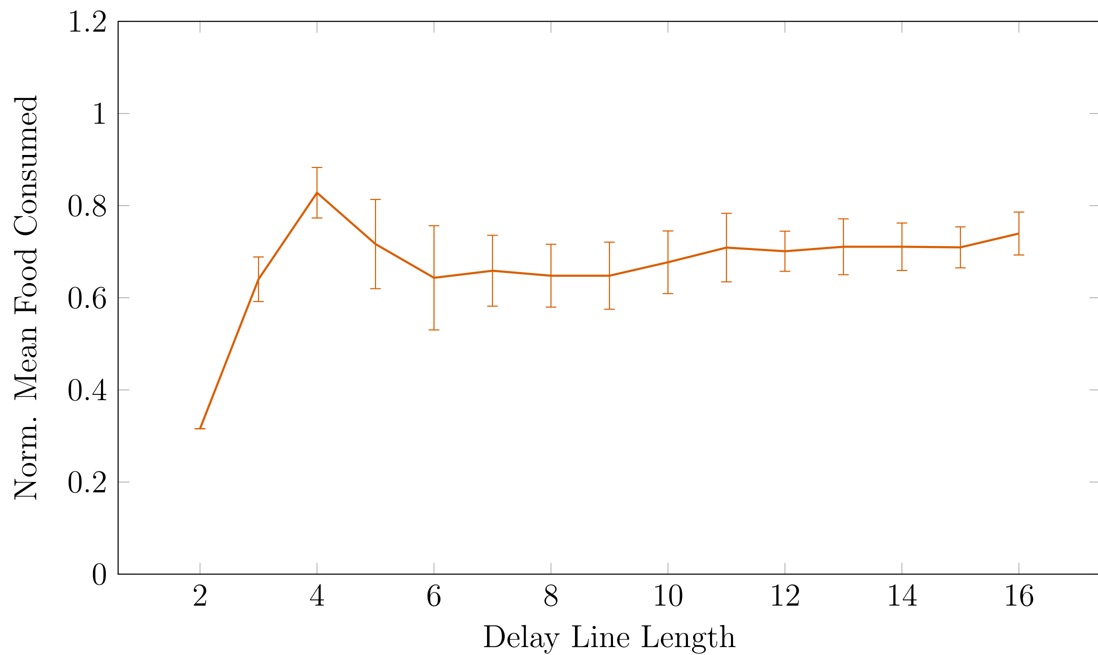


Figure 5.15: Plot showing the normalized food gathered with the average of the best individual across each GA run on the medium trail segment. Values are normalized by dividing the food gathered by the maximum amount on the trail. There is an overall maximum at length four for this segment with fairly consisent standard deviations.

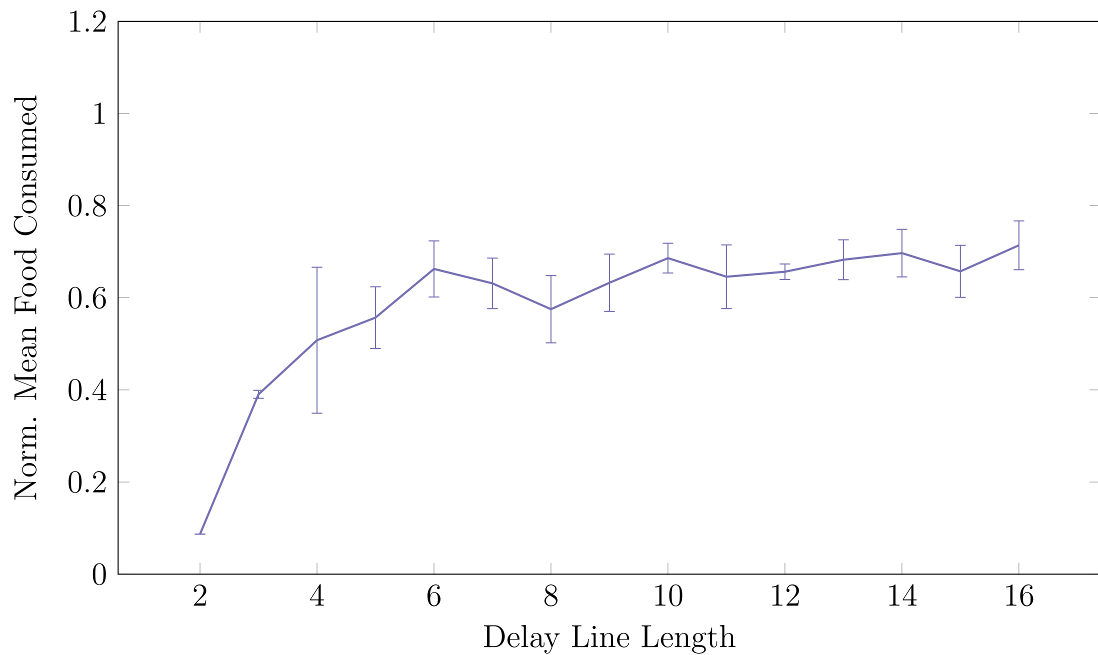


Figure 5.16: Plot showing the normalized food gathered with the average of the best individual across each GA run on the hard trail segment. Values are normalized by dividing the food gathered by the maximum amount on the trail. There is a local maximum around 6, 10, and 16 for these lengths. Note that wide standard deviation through for the majority of these values, in particular at length 4.

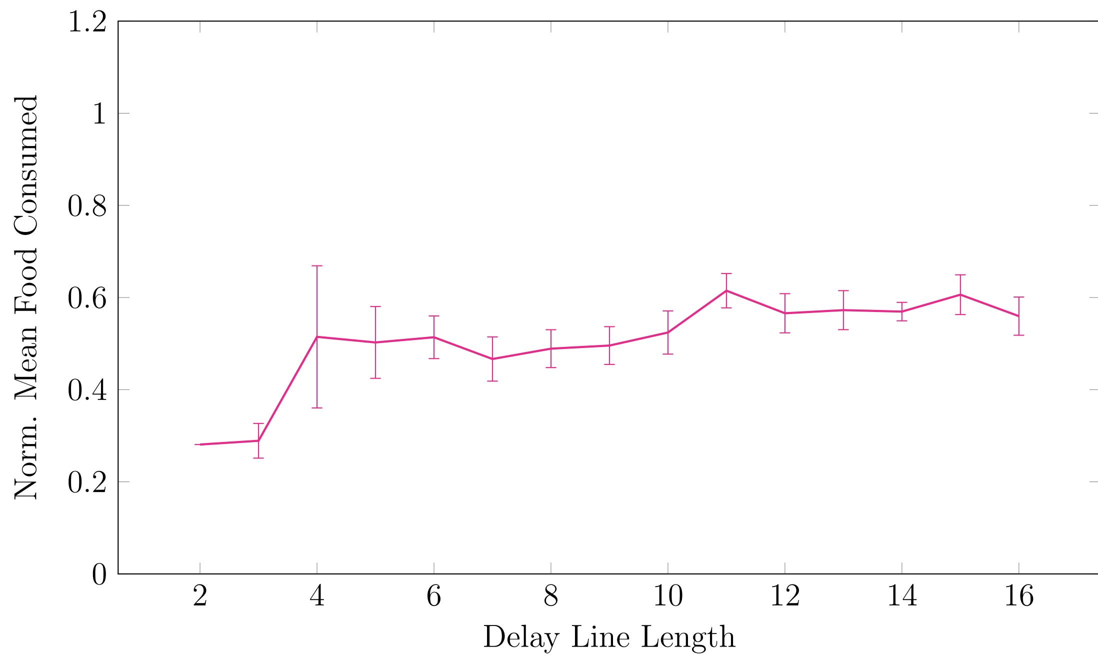


Figure 5.17: Plot showing the normalized food gathered with the average of the best individual across each GA run on the full Santa Fe trail. Values are normalized by dividing the food gathered by the maximum amount on the trail. There is a local maximum around delay line length 4, 6, 11, and 13 in this figure. This chart also shows a wider standard deviation for a delaly line of length 4, as seen in the maximums for this trail (Figure 5.13).

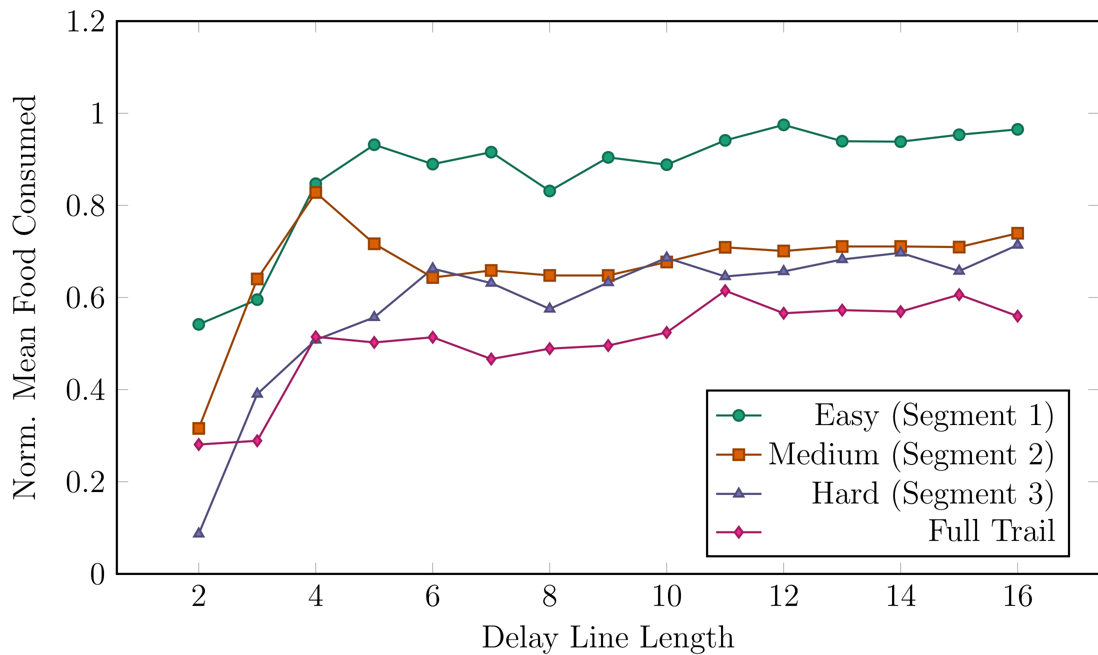


Figure 5.18: Plot showing the normalized food gathered with the average of the best individual across each GA run for all trail segments. Values are normalized by dividing the food gathered by the maximum amount on the trail. There is not a consistent local maximum for all the trails, but all trails have a sizable increase in performance from length three to four.

5.3 Discussion

Test trail 1, test trail 2, and the John Muir trail all found a solution consuming all food on the trails through the GA configuration as shown in Figure 5.4, Figure 5.6, and Figure 5.8, respectively. The criteria for a solution in this case was to consume all of the food within the specified number of moves without regard to optimization of number of moves through the trail. As a result, the path the agent took in test trail 2 (Figure 5.7) and the John Muir trail (Figure 5.9) are both valid solutions, but not necessarily the most efficient means through the trail. Because the simplicity of test trail 1, the agent actually found an optimal solution in Figure 5.5.

Figure 5.10 shows an evaluation that was no longer progressing after around generation 90. This evaluation shows how some runs are terminated early in the non-CRN simulations because the population of individuals grows stale. Even with further mutation and crossover, which is occurring in this diagram, there is no longer variation in the population after 300 generations. As such, this particular run as well many of the others used in our testing were terminated in a similar fashion.

The benchmark on the Santa Fe trail also show the importance of SCOOP in the evaluation of these tasks. Without some form of parallelism, the evaluation of the GA (as expected) takes more than three times as long with 400 moves on the Santa Fe trail (see Figure 5.11). There is an upper bound on the amount of gain which is practically the population size. The maximum number of parallel tasks we could ever have on this task is the size of the population. This is because the fitnesses of the entire population and offspring must be evaluated prior to entering the selection phase.

Figure 5.12 confirms that removal of the no move option was not an issue for

the ANN. Jefferson and Koza both observed the same behavior [12] [24]. Like this chart, they both found that the elite individuals at the end of a simulation would have never used the no move option. A second common theme here is this individual made no right turns. Very often, the elite individuals would eventually evolve to only make left or right turns throughout the trail. As a full example of this behavior on the trail, Figure 5.9 shows a different individual who evolved to only make right turns throughout the John Muir trail.

The next chart, Figure 5.13 shows how we arrived at the desired length of delay line for implementation in a CRN. All four trails agree that a delay line of length two or three is insufficient to solve the task. At length four, the easy trail hits the first point that it is capable of consuming all the food. In addition, the hard trail and the medium trail also make a large jump between the values on a delay line of length three. The same jump occurs on the full Santa Fe trail for the length of the delay line. The objective is to find the *least complex* trail that will consume the maximum amount of food. From the maximum chart alone, the conclusion that a length four is sufficient is drawn for the easy, hard, and full Santa Fe trail. The medium trail requires more investigation because at first glance, it appears a delay line of length six is the best choice.

Notice the medium trail's best performance, on average (see Figure 5.15), occurs when the delay line length equals four. This is confirmed looking at the standard deviation showing that the deviation is lower compared to that of a delay line length six. With this data, the conclusion that a delay line of length equals four, on average, will perform better than a delay line of length six on the medium trail. Using the mean and standard deviation, the results conclude that a delay line length of four is sufficient across all trails to consume the most amount of

food from these results. In the next chapter, we now take this delay line length of $N = 4$ and move it into a CRN.

Chapter 6

Chemical Reaction Network Trail Simulations

In this chapter, we discuss how we integrated the delay line with an Artificial Neural Network (ANN) to solve the navigation of an agent through modified versions of the Santa Fe trails in a Chemical Reaction Network (CRN). We carry over the Genetic Algorithm (GA) shown in the previous chapter and the minimal delay line length of $N = 4$ to implement them as a CRN. We will also show in here that the increased run time of a CRN versus a non-CRN makes the optimization prior to going to a CRN an important step.

This chapter starts out by discussing the methodology we used to gather the data, presents the results, and then provides a discussion on the data presented.

6.1 Methodology

After finding the minimal length of delay line in the non-CRN simulations, we moved with this length of $N = 4$ into the CRN simulations. We performed these simulations represented in a CRN with COllective cELlular computing (COEL) [63]. Each trail and ANN was evaluated 10 or more times in COEL. The CRN ANN was constructed with a network similar to the one in Figure 5.3 with a few differences.

Figure 6.1 shows an example of this modified neural network for the chemistry simulations. The node model we use in the CRN does not require the inverted input so there is a single input node for each delay line stage. Another exception is the presence of a hidden perceptron. We wanted to evaluate performance to see if the hidden perceptron was necessary in the ANN in Figure 5.3. We evaluated

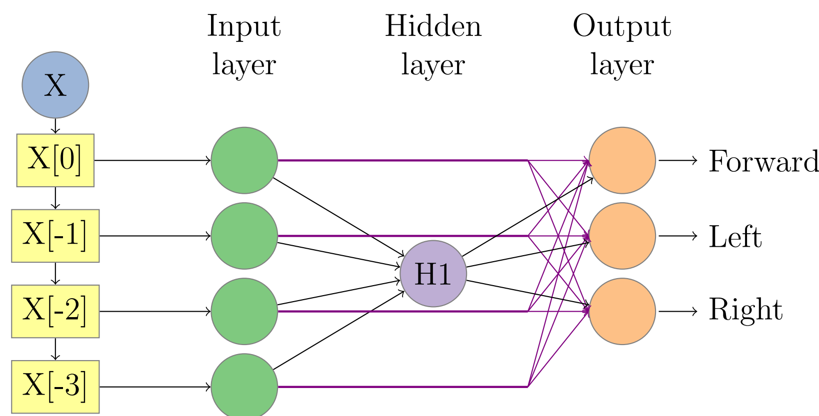


Figure 6.1: This figure shows the ANN combined with a 4-input ($N = 4$) delay line (on left). This feedforward ANN has full connections from the input to the hidden layer, input to the output layer, and hidden to the output layer. The delay line has a single input neuron for each stage in a chemistry. Values shift down the delay line where $X[0]$ represents the current input, $X[-1]$ represents the previous input, and so on.

performance both with the presence of a hidden layer (a single perceptron) and without a hidden layer completely.

The perceptrons were modeled with Banda's Analog Asymmetric Signal Perceptron (AASP) [13] and the ANN in a chemistry was modeled after Blout's compartmental chemistries [14]. Using these two systems allowed us to construct a network that is comparable to the one in Figure 5.3 and the current state of the art in a CRN. Further simplification from Figure 5.3 is possible because there is a single connection from the delay line to the input layer, so we remove the formal perceptron layer and have the delay line itself act as the input layer as shown in Figure 6.2. Each of the nodes in this diagram are represented with a four or five input AASP. Table 6.1 shows the reactions and rate constants of the AASP used to model these perceptrons. The Manual copy Delay Line (MDL) reactions and rates are extended as discussed in Chapter 3 and are shown earlier in Table 3.5. With the modified input layer, this means the total number of nodes in the CRN

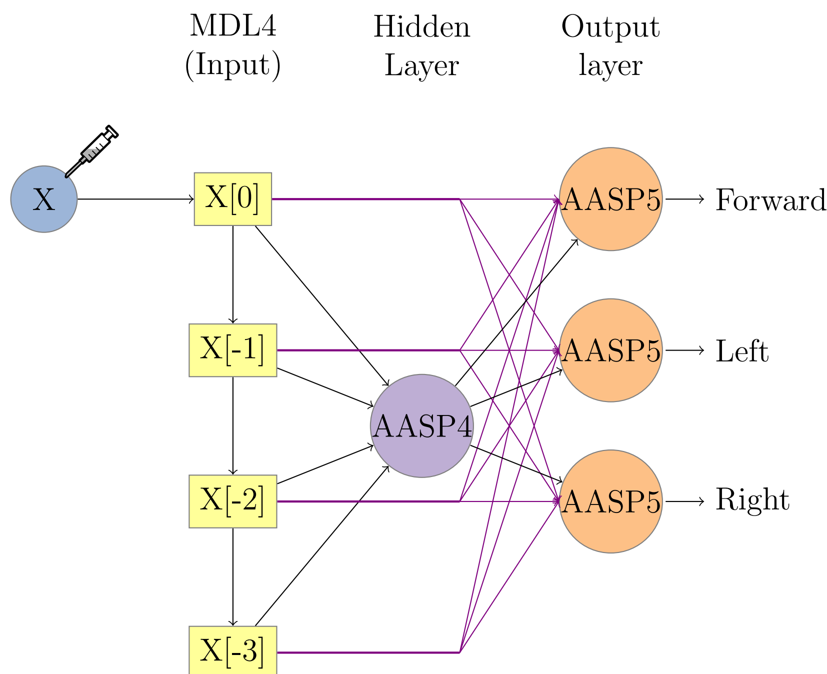


Figure 6.2: This figure shows the actual model of the neural network simulated in a chemistry. The evolved parameters were the weights indicated by each individual between nodes. The AASP4 and AASP5 are the Banda's AASP and the left input column of $X[n]$ boxes represent the length 4 MDL. The syringe on X is the input to the system. Compared to Figure 6.1, the single connection between the delay line and input layer allows us to directly connect to the two in our CRN.

ANN is four with a hidden neuron and three without a hidden neuron.

We chose to represent the delay line in a CRN with a MDL. We did not simulate the system with a Backwards signal Propagation delay Line (BPL) because the average error for a BPL of over two stages is significant compared to a MDL of the delay length (see Section 3.3.2). The additional complication of having to manually signal copy between stages of the MDL was not a concern for our testing. We were not constrained with the number of inputs and outputs we can have, like other chemical systems.

After finding the best set of CRN ANN structures, we wanted to find if the resulting ANN for that particular trail is specialized or is a generalized solution to

these types of tasks. We take the best individual from each trail and then evaluate it on the three other trails as well as Jefferson's John Muir trail (see Figure 2.1) and measure the food consumed. As an example, for the easy trail, we take that individual and run it on the medium, hard, full Santa Fe trail, and full John Muir trail to compare the results. The John Muir trail was limited to 200 moves to match Jefferson's evaluations on the trail where he successfully evolved an ANN that consumed 89 pieces [12].

Reaction	**Rate**	$\boldsymbol{K_m}$	**Reaction**	**Rate**	$\boldsymbol{K_m}$
$S_{in} + Y \rightarrow$	0.1800	(None)	$X_3 + Y \rightarrow$	0.3905	(None)
$S_{in} \xrightarrow{W_0} S_{in}Y + Y$	2.5336	0.5521	$X_3 \xrightarrow{W_3} X_3Y + Y$	0.1227	0.4358
$X_1 + Y \rightarrow$	0.3905	(None)	$W^{\ominus} \xrightarrow{X_3Y} W_3^{\ominus}$	1.6788	0.1889
$X_1 \xrightarrow{W_1} X_1Y + Y$	0.1227	0.4358	$W_3 + W_3^{\ominus} \rightarrow$	0.2416	(None)
$X_2 + Y \rightarrow$	0.3905	(None)	$W^{\oplus} \xrightarrow{X_3Y} W_3$	5.0000	0.2744
$X_2 \xrightarrow{W_2} X_2Y + Y$	0.1227	0.4358	$X_4 + Y \rightarrow$	0.3905	(None)
$T \xrightarrow{S_L} E^{\oplus}$	1.9613	0.1155	$X_4 \xrightarrow{W_4} X_4Y + Y$	0.1227	0.4358
$Y \xrightarrow{S_L} E^{\ominus}$	1.9613	0.1155	$W^{\ominus} \xrightarrow{X_4Y} W_4^{\ominus}$	1.6788	0.1889
$T + Y \rightarrow$	5.0000	(None)	$W_4 + W_4^{\ominus} \rightarrow$	0.2416	(None)
$W^{\ominus} \xrightarrow{S_{in}Y} W_0^{\ominus}$	1.6697	0.6000	$W^{\oplus} \xrightarrow{X_4Y} W_4$	5.0000	0.2744
$W_0 + W_0^{\ominus} \rightarrow$	0.2642	(None)	$X_5 + Y \rightarrow$	0.3905	(None)
$W^{\oplus} \xrightarrow{S_{in}Y} W_0$	2.9078	0.5023	$X_5 \xrightarrow{W_5} X_5Y + Y$	0.1227	0.4358
$W^{\ominus} \xrightarrow{X_1Y} W_1^{\ominus}$	1.6788	0.1889	$W^{\ominus} \xrightarrow{X_5Y} W_5^{\ominus}$	1.6788	0.1889
$W_1 + W_1^{\ominus} \rightarrow$	0.2416	(None)	$W_5 + W_5^{\ominus} \rightarrow$	0.2416	(None)
$W^{\oplus} \xrightarrow{X_1Y} W_1$	5.0000	0.2744	$W^{\oplus} \xrightarrow{X_5Y} W_5$	5.0000	0.2744
$W^{\ominus} \xrightarrow{X_2Y} W_2^{\ominus}$	1.6788	0.1889	$B \xrightarrow{E^{\oplus}} E^{\oplus} + W^{\oplus}$	1.0000	1.0000
$W_2 + W_2^{\ominus} \rightarrow$	0.2416	(None)	$B \xrightarrow{E^{\ominus}} E^{\ominus} + W^{\ominus}$	1.0000	1.0000
$W^{\oplus} \xrightarrow{X_2Y} W_2$	5.0000	0.2744	$E^{\ominus} + E^{\oplus} \rightarrow$	5.0000	(None)
$Y \xrightarrow{S_F} F$	0.1000	3.0000			

Table 6.1: This table shows the reactions and rate constants for the AASP. The "Rate" column shows the forward reaction rate and "K_m" shows the catalyst rate. The catalyst is the species shown above the arrow. This table shows an AASP with five inputs. Removing the reactions containing W_5 and X_5 make this a four input AASP. Notice how each input (X_n) is "weighed" like a classical perceptron through varying concentrations of weights (W_n). Each W_n is the concentration that was set to a random starting value and then varied during the GA process in the CRN. These reactions are based off work from Banda and Teuscher [75].

6.2 Results

In the previous chapter, we arrived at the conclusion that a delay line of length $N = 4$ is sufficient for this task. We mention that the segments were used because they ran faster than the full Santa Fe trail. Figure 6.3 show the run time for a selected run for all three segments and then run time for the full Santa Fe trail in a CRN with a MDL of length $N = 4$.

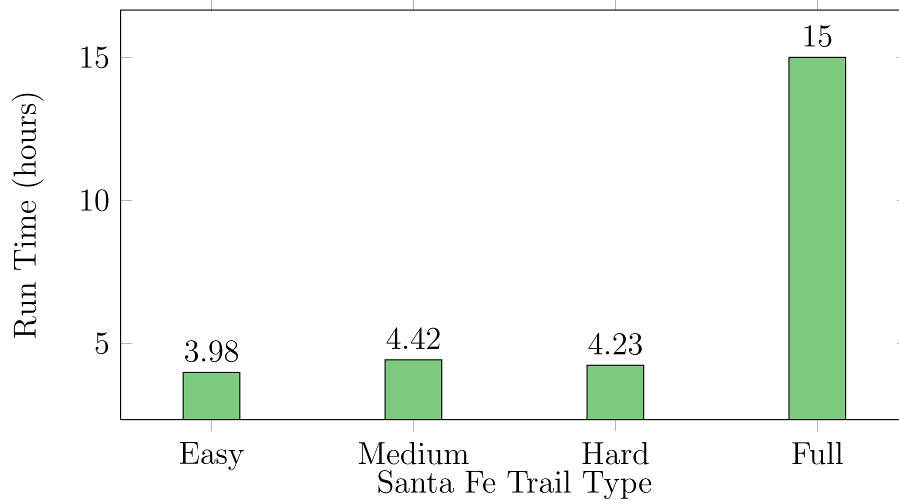


Figure 6.3: Plot showing the run time for each of the trail segments for a full run of 100 generations. Notice how the 400 maximum number of moves causes the full Santa Fe trail to take a much longer time than the four other trail segments that only have a maximum moves of 100.

We also show a table of the first ten generations of a full Santa Fe trail run in Figure 6.4 and Table 6.2. Each run was executed with the same GA parameters and same CRN. The only difference was the changing trails and the full Santa Fe trail had a maximum moves of 400 compared to the typical 100 for each of the segments. These 300 additional moves cause the time for the full trail execution to be approximately 3.5 times longer than the three trail segments.

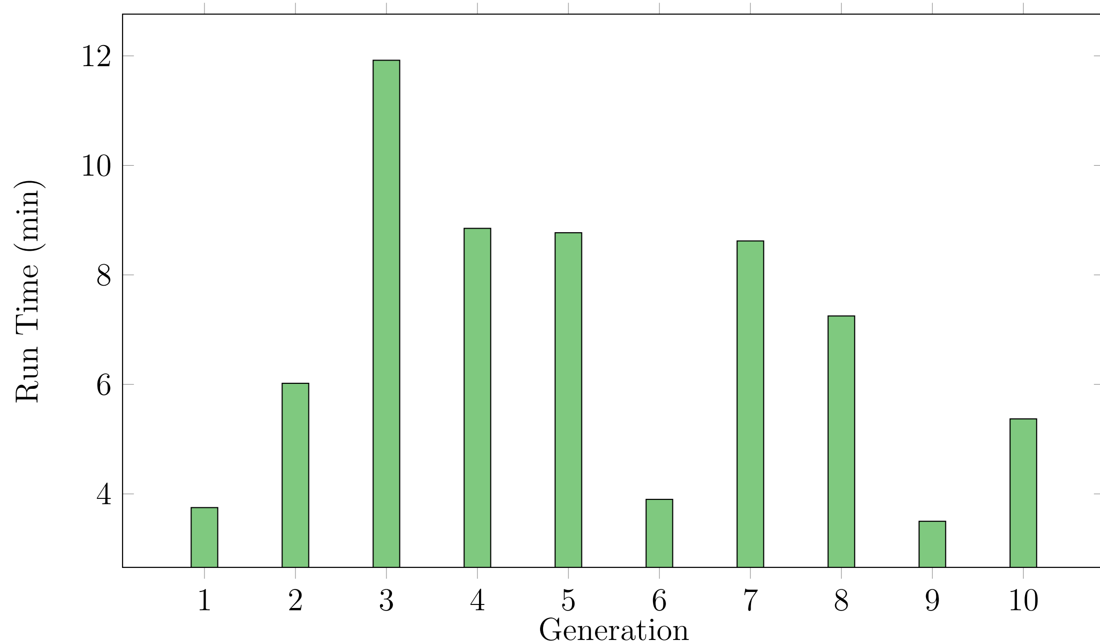


Figure 6.4: Plot showing the run time for the first 10 generations of a full Santa Fe trail run with GA configuration similar to non-CRN configuration. These results are itemized in Table 6.2.

Generation	**Run Time ((HH:)MM:SS)**
1	03:45
2	06:01
3	11:55
4	08:51
5	08:46
6	03:54
7	08:37
8	07:15
9	03:30
10	05:22
Mean First 10 Generations	06:48
Total First 10 Generations	01:07:56
Mean 100 Generations	08:45
Total 100 Generations	14:42:56

Table 6.2: Table showing generation run times for COEL CRN on the first 10 generations and summary of full run of Santa Fe trail with similar non-CRN GA configuration. Compare table to results of Figure 5.11 to see that an entire run can finish in a single generation of the CRN simulation. The individual generations are plotted in Figure 6.4.

Now, presenting a delay line length of $N = 4$ in a CRN. We executed tests both with a hidden neuron and without a hidden neuron once implemented in a CRN to see if the hidden neuron was a necessity for solving this task. The next series of charts show the food consumed for each version of the trail and compares it to Koza's original results and the results from the non-CRN come from Figure 5.13 where MDL length is 4. Figure 6.5 shows the maximum individual performance of the trail versus the non-CRN implementation for each trail segment and the full Santa Fe trail. Figure 6.6 show the mean and standard deviation. These charts are normalized using the maximum food available on a given trail and both composed of 10 or more CRN runs.

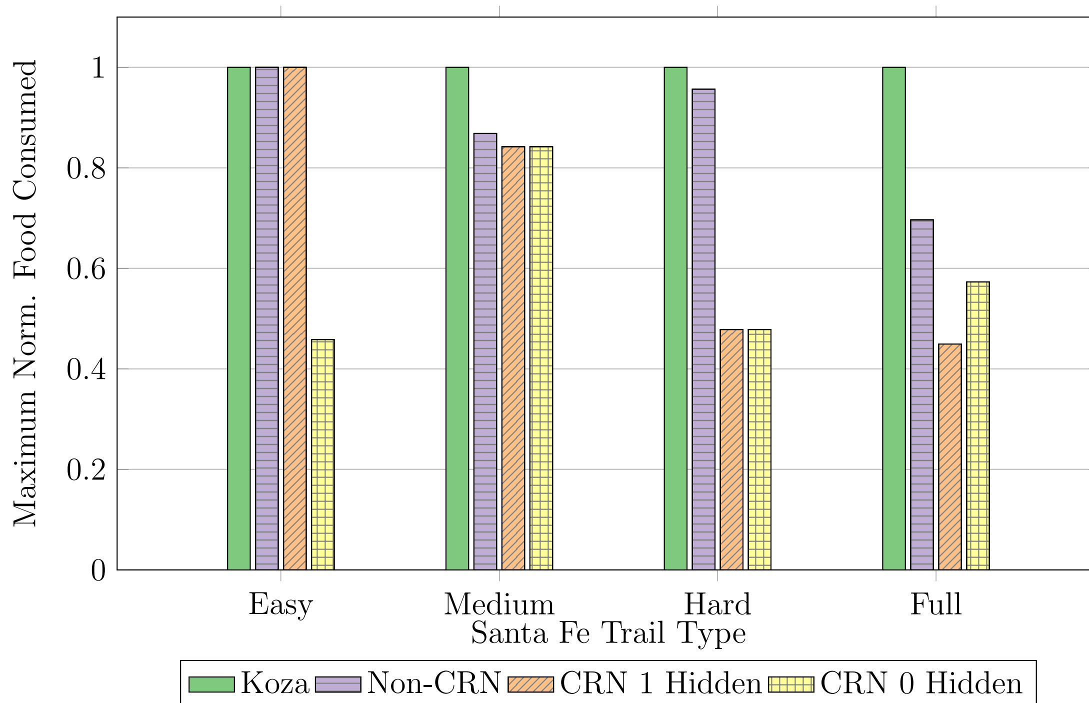


Figure 6.5: Plot showing the normalized maximum food consumed for each trail and each ANN configuration on the Santa Fe trail. The maximum food obtained for each trail seems meet or exceed the performance of that with a hidden layer with the exception of the easy trail.

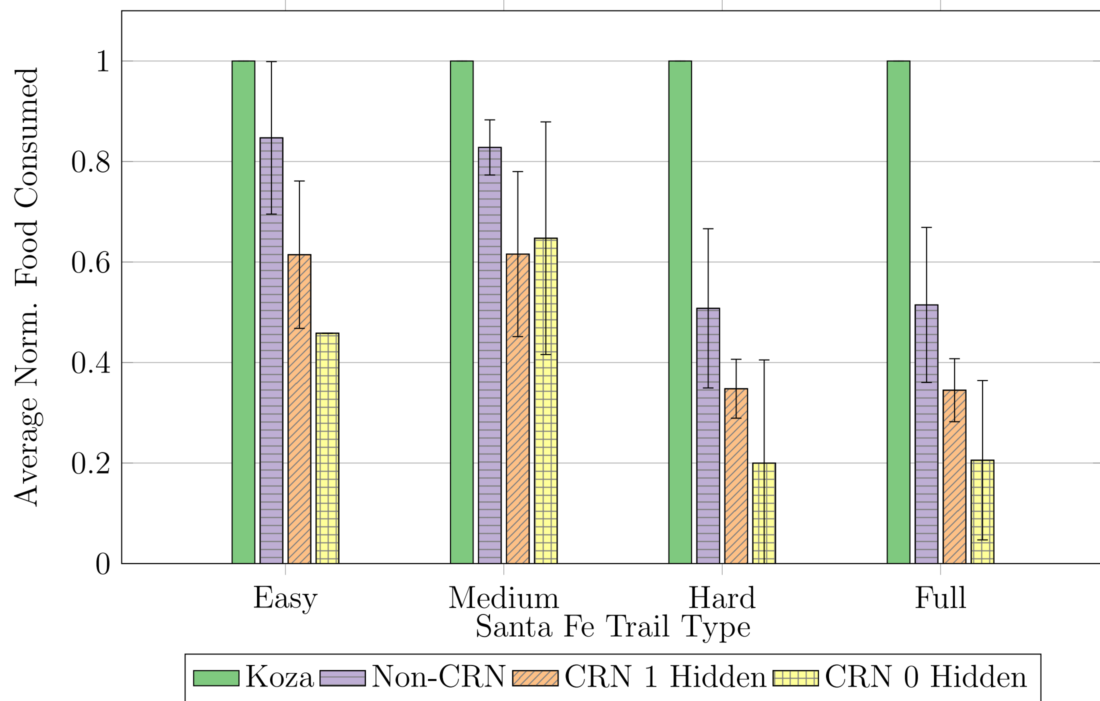


Figure 6.6: Plot showing the normalized average food consumed for each trail and each ANN configuration on the Santa Fe trail. Error bars are standard deviation. It appears, on average, that a hidden layer helps the system find more food for each run. The removal of the hidden layer also seems to produce a wider set of possible values versus the hidden layer having a tighter standard deviation which implies more consistent results.

We now show the probability of finding food on each of the trail segments that tests were ran against. Figure 6.7 takes the pieces of food on each trail and divides it by the total number of squares on the trail. For the segments, there are 256 squares and the full Santa Fe trail contains 1024 squares. The John Muir trail with the same pieces of food in the same area has the same probability as the full Santa Fe trail.

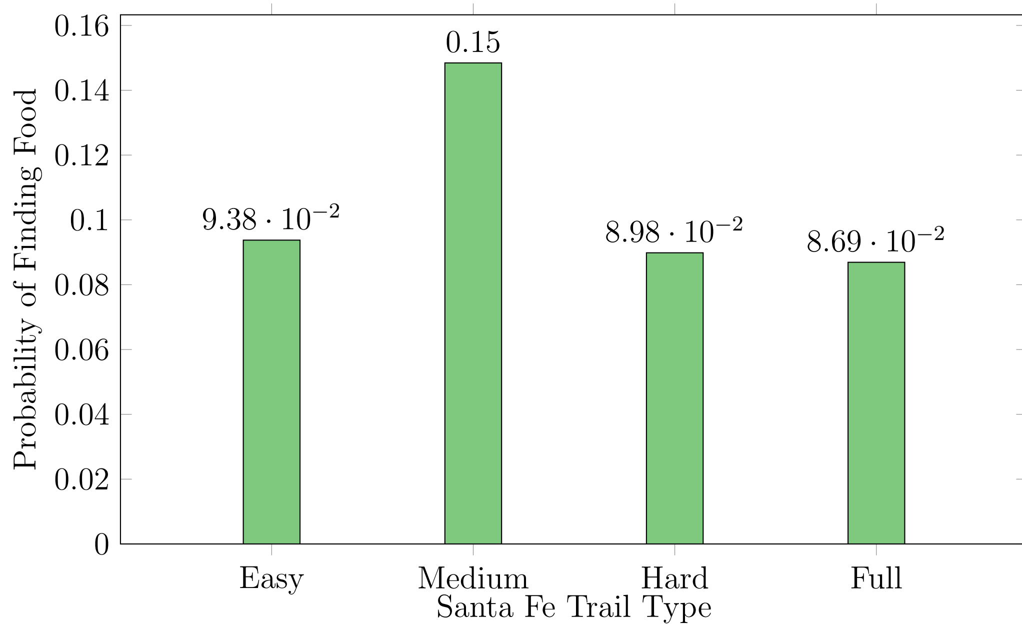


Figure 6.7: Plot showing the probability of finding food on each trail. This is calculated by taking the maximum amount of food on each trail and dividing it by the total number of squares on each trail. Notice how the full Santa Fe trail has the lowest probability of randomly finding food and the medium has almost twice the probability of randomly finding food compared to all three other trails.

Figure 6.8 and Figure 6.9 show the percentage error calculated using two different methods. The first is is the percent error from the maximum food available. The second percent error is the percent error from the CRN results. The results for the non-CRN simulations and the CRN simulations are summarized in Table 6.3.

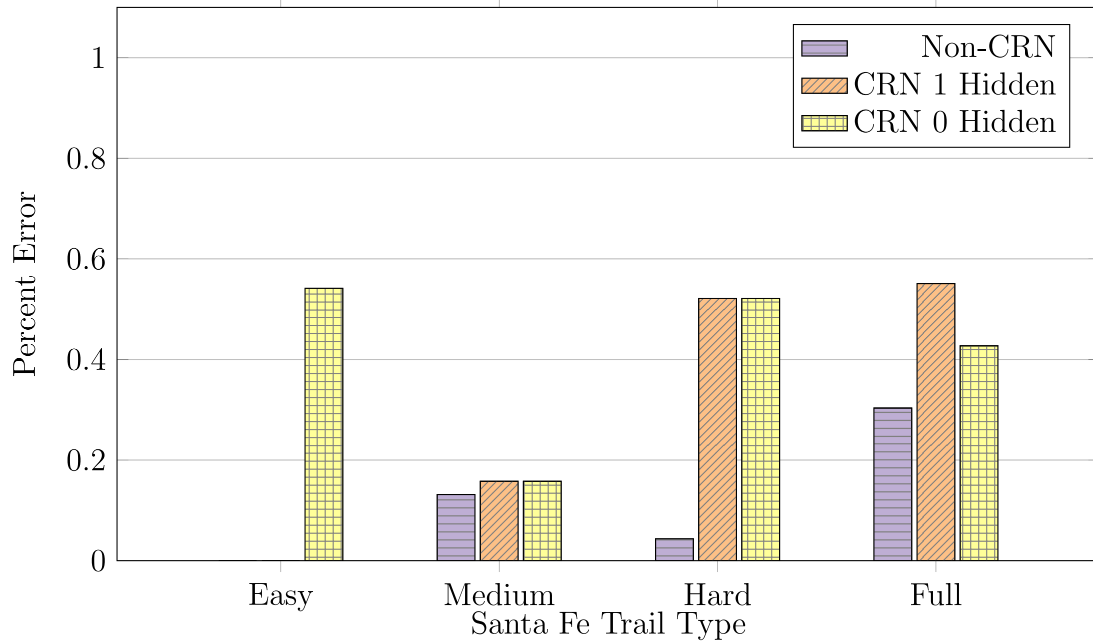


Figure 6.8: Plot showing the percent error from maximum amount of food available on each trail segment. Even the the non-CRN implementation struggles with some of the trails with larger amounts of food like the medium and full trails.

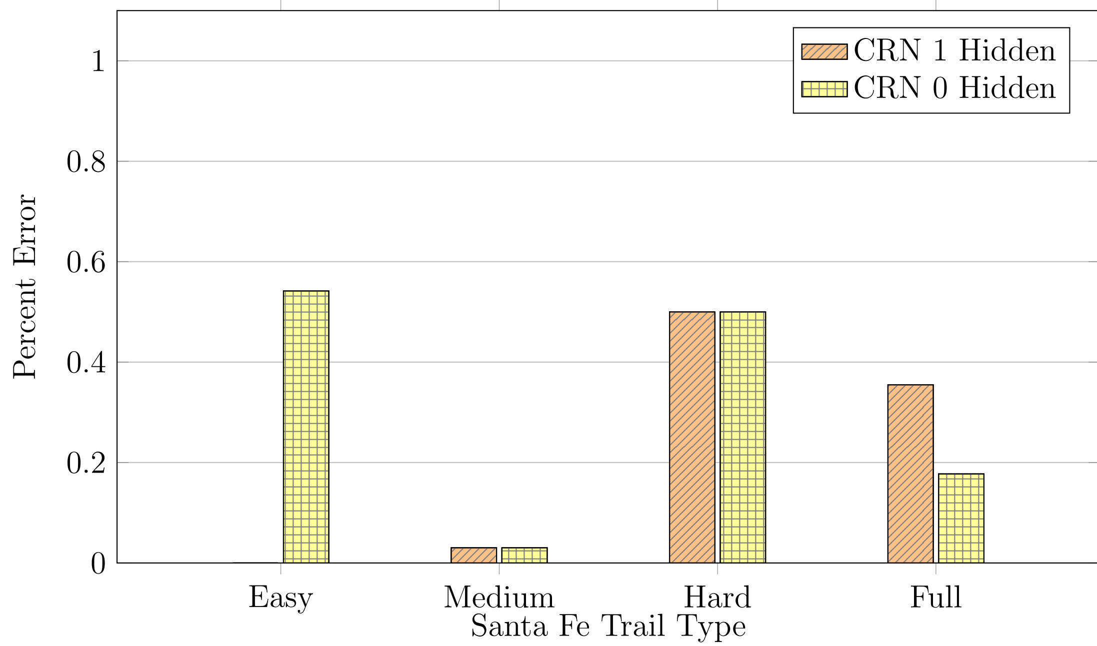


Figure 6.9: Plot showing the percent error from the maximum performance of the non-CRN. On the segments, the CRN version with a hidden layer seems to beat or perform at the same level as the version without a hidden layer. For the full trail, it seems that the hidden layer does not provide an advantage.

Trail		Koza	Non-CRN	CRN	CRN No Hidden
Easy	Max	24	24	24	11
	Mean	24.00	20.33	14.75	11.00
	Std. Dev	0.00	3.64	3.52	0.00
	% Error (Max)	n/a	0.00%	0.00%	54.17%
	% Error (from CRN)	n/a	n/a	0.00%	54.17%
Medium	Max	38	33	32	32
	Mean	38.00	31.47	23.40	24.60
	Std. Dev	0.00	2.09	6.24	8.80
	% Error (Max)	n/a	13.16%	15.79%	15.79%
	% Error (from CRN)	n/a	n/a	3.03%	3.03%
Hard	Max	23	22	11	11
	Mean	23.00	11.68	8.00	4.60
	Std. Dev	0.00	3.64	1.35	4.72
	% Error (Max)	n/a	4.35%	52.17%	52.17%
	% Error (from CRN)	n/a	n/a	50.00%	50.00%
Full Trail	Max	89	62	40	51
	Mean	89.00	45.80	30.70	18.30
	Std. Dev	0.00	13.73	5.58	14.11
	% Error (Max)	n/a	30.34%	55.06%	42.70%
	% Error (from CRN)	n/a	n/a	35.48%	17.74%

Table 6.3: Table showing the summary of food consumed for each trail and each network type. The percent error is calculated in two parts. The first is from the total amount of food available in each segment and the second is the percent error from the CRN configurations. The CRN with a hidden layer ("CRN") performs better or the same as each trail than the no hidden layer configuration except for the full trail. The lack of a hidden layer on the easy trail negatively affects the agent's ability to gather food.

Figure 6.10 shows an evaluation of taking the best individual from each evaluation in the CRN with a single hidden neuron and evaluating that individual on the other trails. Figure 6.11 shows the same without the hidden neuron.

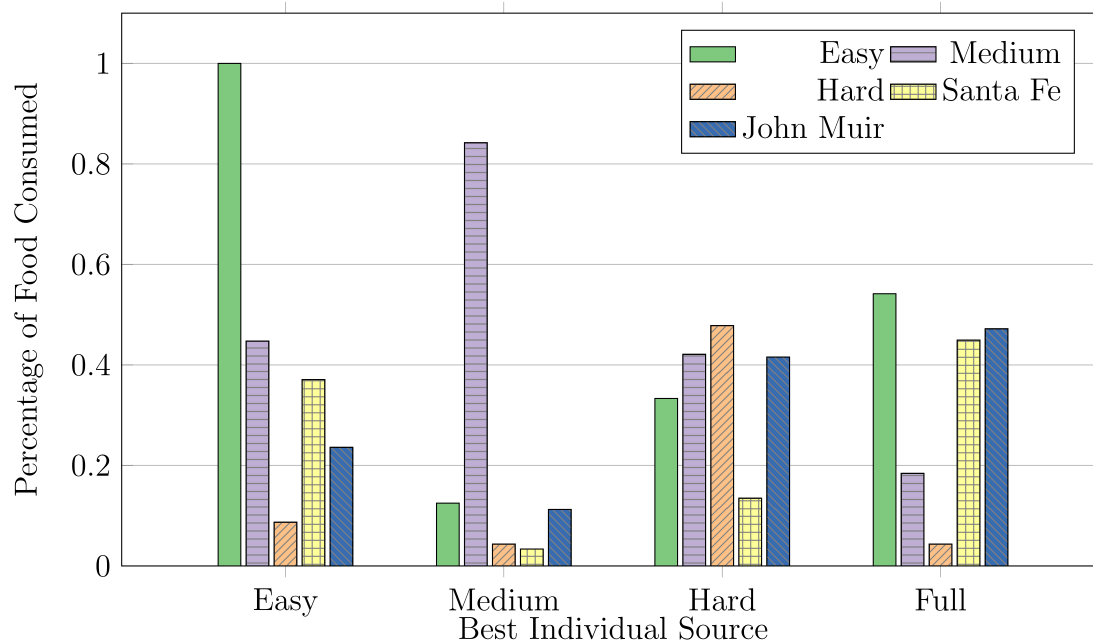


Figure 6.10: Plot comparing the best individuals performance from each trail against evaluation on other trails for the ANN in a CRN with one hidden perceptron. Each group of bars, such as “Easy” on x-axis, correspond to the same individual ran on a different trail.

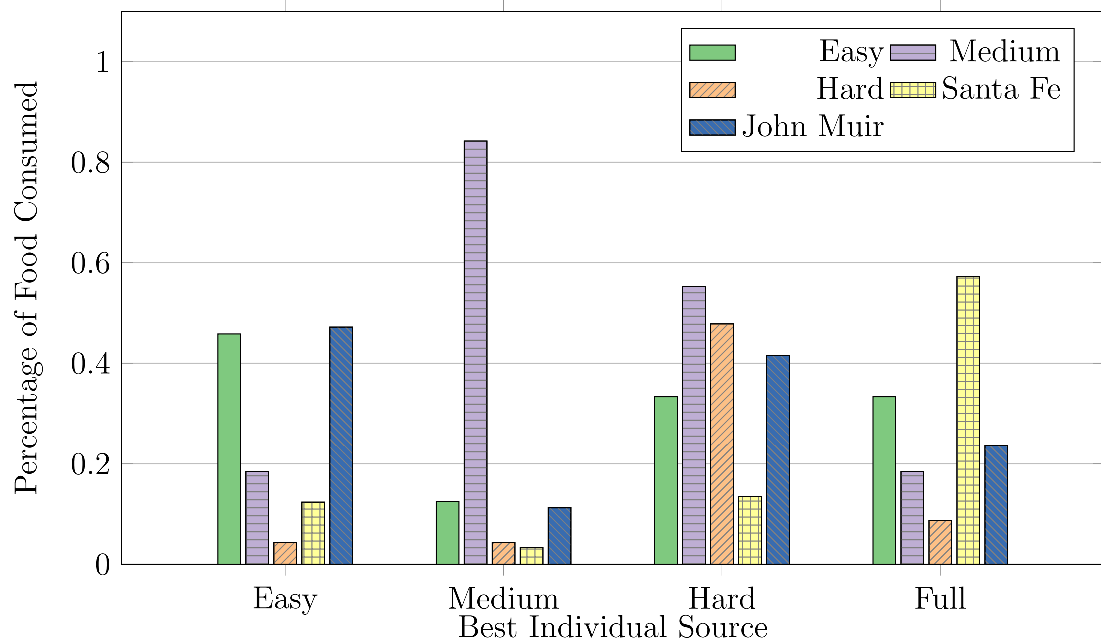


Figure 6.11: Plot comparing the best individuals performance from each trail against evaluation on other trails for the ANN in a CRN without a hidden layer. Each group of bars, such as “Easy” on x-axis, correspond to the same individual ran on a different trail.

Figure 6.12 and Figure 6.13 show histograms of the count of evolution runs that consumed each amount of food.

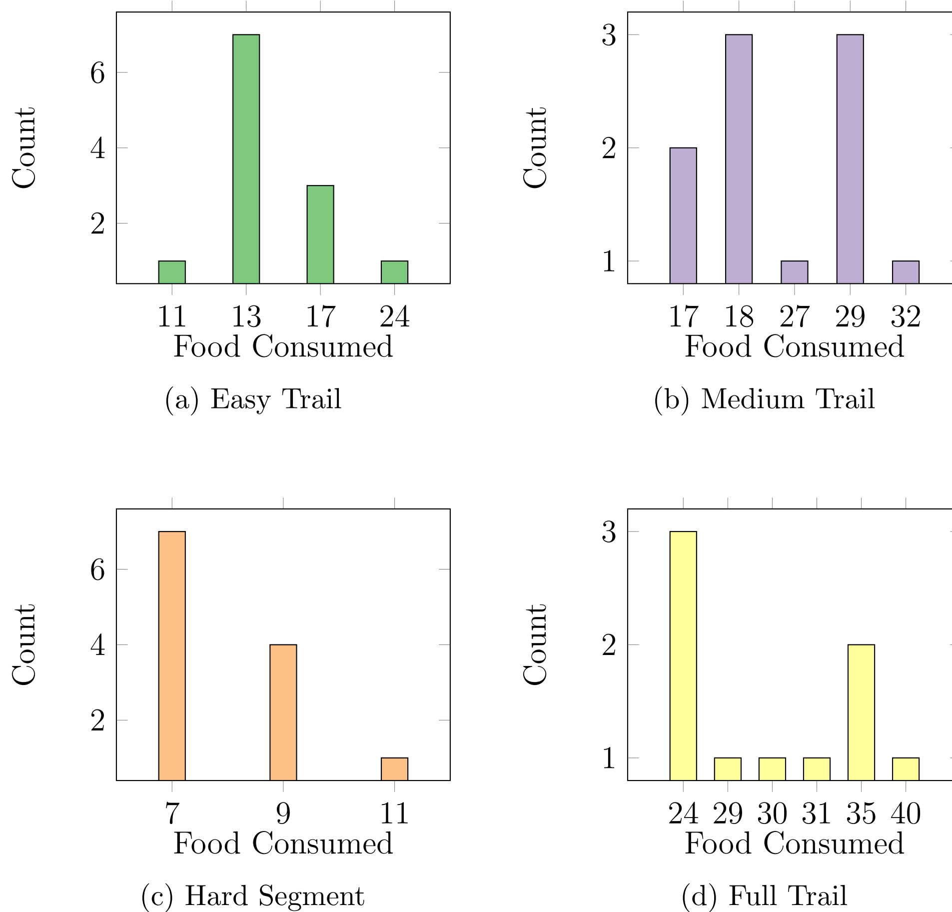


Figure 6.12: Set of charts showing the number of evolution runs with elite individuals from CRN with a hidden layer collecting each amount of food. The top performers on each trail are only one individual, but there are other values not far below the top performer. Out of all the runs on each trail, only one GA run lead to the top performer.

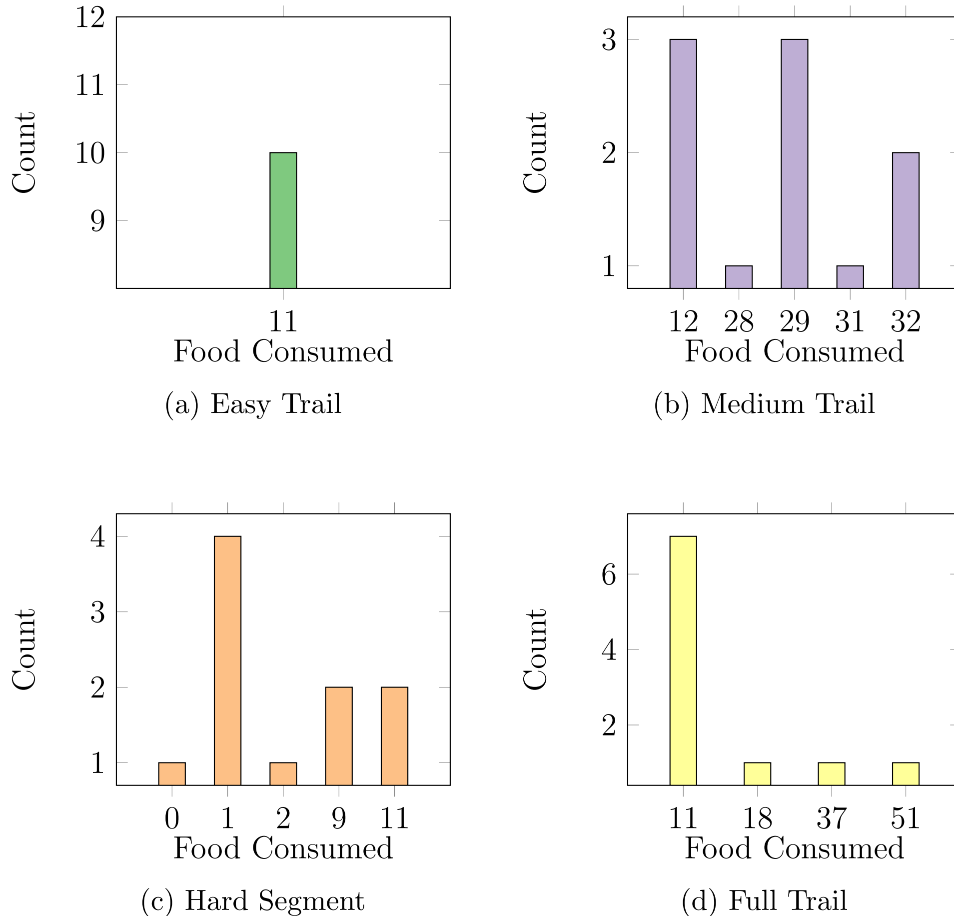


Figure 6.13: Set of charts showing the number of evolution runs with elite individuals from CRN without a hidden layer collecting each amount of food. For easy, no GA run lead to varying performance. On medium and hard, more than a single GA evaluation lead to a top performing individual and the full trail has a spread of values with three outliers above the typical food consumed of 11. This suggests further GA refinement may improve the results.

6.3 Discussion

The timing evaluation in Figure 6.3 and Table 6.2 shows the importance of the preliminary research of these simulations in the non-CRN environment. One fact we mentioned earlier in this chapter is the long run time that the CRN simulations can take compared to the time it takes to perform a similar simulation in a non-CRN environment. A single generation of a run in a CRN takes longer than an entire run of 100 generations in trail runner. Even the fastest generation in the CRN simulation took more than one and a half times longer than the tests with the non-CRN tools. We can determine an approximate time frame it would have taken to perform this same optimization in a CRN.

As an example calculation, assume that we take the minimum number of GA evaluations for each the three trail segment, 70 runs, and the full Santa Fe trail, 25 runs. This is performed across 15 delay line lengths from 2 all the way up to 16. We can approximate the total run time by using the minimum time for all three segments (239 minutes) to arrive at $15 \times (70 \times 3 \times (239) + 25 \times (902)) = 1,091,100$ minutes. This is over two years. If we assume that we could run ten of these jobs in parallel, this still results in around eleven weeks to complete the same set of simulations we accomplished in a fraction of the time by evaluating the GA performance in trail runner prior to moving the networks to a CRN.

We now take the delay line of length four and look at the results in a CRN. The two charts with these results are the normalized maximum food consumed and the normalized average food consumed in Figure 6.5 and Figure 6.6, respectively. Looking at the maximums first, we get the same performance as the non-CRN ANN only on the easy trail for the CRN ANN with one hidden layer. From the maximums, there is not a clear advantage to the addition of the hidden neuron. In

the medium and hard trails, the performance of no hidden layer is able to match that having a hidden layer. In the case of the full Santa Fe trail, no hidden neuron even outperforms the ANN with a hidden neuron.

For the medium and hard trail, the hidden neuron does not seem to lend an advantage from the maximum values in Figure 6.5. Looking at the means (Figure 6.6), the maximum performance for that particular individual on the hard trail seems to be an outlier compared to the average performance on the trail. With the wide standard deviation on all of the no hidden layer CRN networks, except for the easy, it seems that individuals may perform that well, but vary widely within the tests. For the medium, the average food consumed without a hidden perceptron seems to outperform having the perceptron with only a slightly wider standard deviation. Why is this the case? A potential explanation is the probability of even finding food on the medium trail.

Figure 6.7 shows the probability of finding a food on any of the given trails. Notice that the medium has a probability that is almost twice as large as any of the other trails. This means that a non-optimal individual on the trail has a higher possibility of collecting some amount of food on the trail. With the wide standard deviations for the no hidden layer CRN individuals, it seems that some of the performers in this group could be low performing individuals wandering and finding food on the trail. This may be the case for some of the CRN with a wider standard deviation. That said though, there are other instances for both with and without a hidden layer in a CRN that a random search seems less likely with a tighter mean.

Next, we show the percent error for the non-CRN and both CRN models calculated from Koza's results in Figure 6.8. We see that the non-CRN perform

decently on the three segments, only getting over 20% error on the full trail. Another view to look at this data is using the non-CRN results as the baseline for the percent error and that is shown in Figure 6.9. These results are also summarized in Table 6.3.

Looking at Figure 6.9 and Table 6.3, it seems that for the CRN with a hidden neuron, we achieve a percent error of 50% or less for all of the trails. Excluding the hard and full trail, the CRN with a hidden neuron is able to navigate the trail with less than 10% error. For the CRN ANN without a hidden neuron, the percent error is less than 60% across all trails. At least for the easy trail, where probability of randomly finding food seems less at play, we can conclude that our CRN with a hidden neuron in the ANN has solved a simpler version of this problem. Now, we will examine performance for the best network on other trails.

Figure 6.10 shows the performance of taking the best individual from each ANN and trail evaluation and grading performance on the other trails. Looking at the results with a hidden neuron first, it seems that the strategies evolved are rather specific to each of the trails. The top performer for each group is the individual who was evolved on the trail, with the exception of the full trail. In the full trail, the individual evolved for this trail actually consumes a greater percentage of the trail. This is likely due to the decreased area of the trail: 16×16 in the easy trail versus 32×32 in the full trail. The smaller area means that an agent can wrap around the edge of the trail with fewer moves thus consuming more food in the limited number of moves.

Figure 6.11 shows the same results without a hidden layer. Results for the medium are similar to the CRN with a hidden layer where it performs the best, but this is not the case for the other trails. With the easy, it seems that the agent

consumes all food in front of it and at the first gap, it gets stuck and spins. The percentages of food consumed correspond exactly to the amount of food until the first gap for each trail. The hard results seem more likely that a wandering search is at play. With medium having the highest probability of finding food, it makes sense that wandering the trail for pieces and some reasonable turning strategy would find food. For the full trail, it is not as clear to make a conclusion.

On the full trail evaluation without a hidden layer, it seems that there are a couple possibilities with these results. One is that the agent actually learned a method to solve the trail, but this seems unlikely. If this was the case, we would expect to see greater performance on the easier John Muir trail or the same on the easy segment of the Santa Fe trail. The more likely scenario is there was a wandering agent who got particular lucky on the full Santa Fe trail. This makes sense for the other trails, but this does not seem consistent in the medium trail. For the agent to get as much food as it did on the full Santa Fe trail, it would have had to make turns of some sort when it encountered food or a pattern of food then with food going away. It seems possible that if an agent got caught on a row that contained now food, it may just continue forward until it runs out of moves because it will never consume any more food on the row. Getting stuck on the wrong row seems like a potential explanation for these particular results.

Figure 6.12 and Figure 6.13 show the number of individuals consuming each amount of food for with and without a hidden perceptron. In the results with a hidden peerceptron, we see that only one individual accounted for the top performer on each trail. Others were not far behind of achieving the top performance though with the easy trail being the largest gap. The hidden trail shows similar results with the exception of the full trail. The lack of a hidden neuron neuron

on the hidden trail seems to point more towards a wandering individual scenario where the top two individuals at 37 and 51 pieces of food consumed had a lucky wandering strategy.

In summary, it seems that we have shown that we can partially solve the trails. In the case of the easy trail, we can conclude that we did solve this trail with the CRN with a hidden neuron consuming 100% of the available food. For the medium, hard, and full trails, it is difficult to conclude if there was an individual who actually solved the trail or if there is strategic wandering of the trail leading to optimal values of food consumed. Based off the results, it seems that the addition of a hidden neuron seems to provide a slight edge in terms of strategic food gathering rather than wandering with luck. With the evolution charts presented by Jefferson and Koza, it seems that strategic wandering did occur to an extent early in their algorithms, but was eventually optimized out of the best individual.

Another factor to consider is the GA used on these trails. The GA is similar to the one used on the non-CRN and did not go through the thorough optimization that the non-CRN GA did to arrive at the ideal values. With the difference in how these two systems are implemented in a non-CRN and CRN environment, further optimization of the GA in a CRN would likely lead to better results. As shown in Figure 6.3 and discussed earlier in this section, such an optimization consumes a substantial amount of time with present models for CRNs. In addition, the non-CRN simulations were permitted to run for more than the limited 100 generations we did in a CRN due to computational time. As the models and computational power continue to mature, this optimization may be more practical at a future time.

Chapter 7

Chemical Reaction Network Realization

This chapter discusses the potential implementation of the systems we discuss as a physical, chemical system. It is also possible to design as a sensor that connects to a traditional architecture, like Complementary Metal-Oxide-Semiconductor (CMOS), but we focus on a exclusively chemistry implementation here. We believe that the examples we discuss at the beginning of this work would benefit more as a full chemical system rather than a hybrid or sensor-system approach. We will first talk about various ways to map the system to a wet chemistry and how then fast the system would operate with state of the art.

7.1 Chemical Representation

In Chapter 3 and Chapter 6, we used a set of reactions and species to represent our systems using the models of Michaelis-Menten [58] [59] [60] and mass-action kinetics [56] [57]. Present work has mapped these set of rate reactions to different physical realizations. One such work is by Arkin and Ross, who implement a series of enzymatic gates that correspond to a truth table for a given logic function [3]. Arkin and Ross show that it is possible to implement both a logical AND and OR using a Glycolysis/Gucoeogensis mitocondrial TiCarboxylic Acid (GGTCA) model. Kompa and Levine use different chemical compounds that react at a faster rate to build similar types of logic gates to Arkin and Ross [76].

Another applicable mapping for our work a DeoxyriboNucleic Acid (DNA) strand displacement model from Zhang and Seelig [77]. In this paper, Zhang and

Seelig demonstrate the construction of a DNA walker that is capable of decision making with the use of only proteins. Similarly, Semenov *et al.* [78] showed a more complex version of the walkers (that they called spiders) that have the ability to move along DNA and manipulate or read values. Qian *et al.* builds linear threshold circuits that also operate in DNA strand displacement models to solve logic gates like AND, OR, NOT, and XOR through the use of Artificial Neural Network (ANN)-like structures. All three of these works demonstrate that it is possible to create a mapping of chemical reactions to physical, chemical systems. Using similar principles from these works, we could take our equations from Chapter 6 to a set of DNA strand displacement models that could achieve our desired result.

Stojanovic and Stefanovic have also shown how deoxyribozyme catalysis can be used to solve games like tic-tac-toe using 23 logic gates built at a molecular scale [10] [79]. Their system was capable, in a wet chemistry, of playing successful games of tic-tac-toe with human players using fluorescence as a detection method. Liu *et al.* has also used dexoyribozyme catalysis to implement antibody and nucleic acid detectors [80]. This technology is likely the best candidate of mapping our system to a wet chemical implementation. As a small example, we can show how something like the delay line would look mapping with similar technology to this.

Figure 7.1 shows an example of a length two Manual copy Delay Line (MDL) with the signals being the deoxyribozymes $X1_{signal}$ and $X2_{signal}$, which cleave the substrate X at the embedded ribonucleotide. This produces $X1$ ready for the next system to consume. Subsequently, $X1C$ embedded with another ribonucleotide is able to get cleaved by deoxyribozyme $X2_{signal}$ to form the next input to the system, $X2$. This system is at a similar scale to that discussed in the work by Stojanovic *et al.* Let us now take a look at implementation of a full trail solving

system.

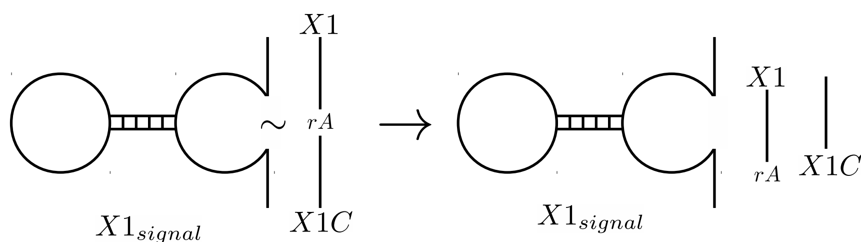


Figure 7.1: Deoxyribozyme cascading example. Deoxyribozyme $X1_{signal}$ cleaves X at embedded ribonucleotide (rA) to form $X1$ and $X1C$. A similar process occurs on $X1C$ to produce $X2$ and $X2C$.

We have discussed works operating typically on only a small region of DNA or representing only a few chemical reactions. The size of the system necessary to construct our full ant solver is much larger than these. For example, take the MDL connected to the perceptrons in the previous chapter to form the ANN for our ant trail system. Each four- and five-input Analog Asymmetric Signal Perceptron (AASP) is composed of 33 and 38 reactions and 37 and 42 unique species, respectively. Recall from Chapter 3, a four-input delay line requires eight reactions and 13 species. So, for an entire trail system with a hidden layer, this results in $8 + 33 + 3 \cdot 38 = 155$ reactions and $13 + 37 + 3 \cdot 42 = 176$ species for our system. This number of reactions and species is greater in complexity than other systems we have found presently implemented in a chemical system.

As an example, the four-neuron Hopfield associative memory from Qian *et al.* contained around 160 reactions with 72 initial DNA species [81]. While this system contained more reactions, the number of species co-existing in this system is less than half the number predicted for full computation of our trail system. One issue that Qian *et al.* discuss in their work is that scaling up the system can amplify leak reactions that occurred and degrade performance of the overall system. Adding

the increased number of species we require could have this type of issue. Another observation by Stojanovic and Stefanovic in their work was erroneous output by gates that should not be signalling. The authors mention that this behavior may lead to errors cascading in larger networks that may lead to undesired behavior [10].

Taking a system, such as this today into a wet chemistry is not impossible, but perhaps difficult based on the current state of the art for the field. It seems that there is not a system of this scale presently implemented as a wet chemistry and other work by Qian *et al.* mentions potential issues as their systems continued to grow in size. One area of improvement is perhaps looking at a way to reduce the complexity of the network in a Chemical Reaction Network (CRN). With more work on complex network implementation in a CRN, in theory, it should be possible to map the trail system reactions with the methods used by Zhang and Seelig, Semenov *et al.*, Stojanovic and Stefanovic, or Liu *et al.* to a physical chemistry. Components of the system, like the delay line by itself, require a smaller number of reactions and may be more feasible for present implementation. If implemented in a wet chemistry, let us now discuss if we can predict the speed at which this system would operate based off current work.

7.2 Processing Speed

As discussed in the previous section, the applicable mapping of our work is likely the deoxyribozyme catalysis [10] [79] [80]. Stojanovic and Stefanovic specifically find that 15 minutes is a reasonable time to cleave and accurately observe the results from the reactions in their tic-tac-toe system. The authors did observe that changing the size of the gates in the chemistry or the concentration of inputs did have an impact on the reaction time, but did not elaborate beyond that. Zhang

and Seelig [77] also observe times in the scale of several minutes to compute similar systems in a DNA strand displacement model.

The advantage of the chemistry is the chemical reactions (for example, each calculation of output layer peceptron in our ANN controlling the ant) can occur in parallel. This of course assumes separation using the compartmental chemistry by Blount *et al.* [14]. As an example, if we take the 15 minute time found by Stojanovic and Stefanovic per reaction, we can predict the cycle time for a single calculation in our trail system.

Let us start with the network in Figure 6.2 and predict the time to complete a movement for the trail system. With a length four MDL, that requires waiting on four steps to occur for shift and store the input value to each block of the delay line. Then, there is a required period of calculation at the hidden neuron followed by one in the output layer. The output layer perceptrons are all able to calculate their results in parallel unlike the delay line, which depends on previous inputs.

So, in total, that gives six stages of calculation. With the 15 minutes found by Stojanovic and Stefanovic, that means approximately 90 minutes for a single calculation step for the full ANN. Running a full system with 200 steps of the John Muir trail would take approximately 12.5 days with this current technology. It is important to note as well that the time we are discussing here is the estimated time to reach a steady state. In a wet chemistry, all of the reactions are occurring in parallel, so a separation between the nodes represented in our ANN diagram is critical.

Catalysts or other signaling methods are required to prevent the later nodes from consuming the species too early. For example in the delay line, we use our Xn_{signal} species to perform this separation. Also, unlike in an electrical system,

the inputs are consumed, meaning, that as we perform the calculation, we are actually decreasing the value of the input as the calculation occurs.

Even though the value is over 12 days for this example, other architectures in chemistry may provide a faster run time. Kompa and Levine [76] discuss the use of different chemicals like aberchromes (I), fulgides, and merocyanines (II) that show dynamics faster than discussed from Stojanovic and Stefanovic. The authors even predict that with continued development of their work, it may be possible to see chemical reactions with photophysicochemical processes operating on a picosecond range.

Chapter 8

Conclusion

In this work, we have shown the feasibility of using Chemical Reaction Networks (CRNs) as a means to implement control systems. We have demonstrated a partial solving of the Santa Fe trail in a CRN. We have also shown the ability of a CRN to partially navigate three sub-segments of the Santa Fe trail. We successfully navigate the easy Santa Fe trail easy segment consuming 100% of the food. For the medium and hard segments and full Santa Fe trail, we are able to consume more than 44% of the available food. This shows that it is possible to solve simpler versions of the trail without a need for Artificial Neural Network (ANN) recurrence. Koza's genetic programs were able to consume all of the food on the Santa Fe trail. With further Genetic Algorithm (GA) optimization, ANNs in a CRN capable of consuming all food seems plausible.

We have also designed a flexible size memory necessary to provide storage for such control systems. The integration of our Manual copy Delay Line (MDL) with the trail system ANN and Asymmetric Signal Perceptron (ASP) demonstrate the ability to hold values over time for later consumption by another system. With the Backwards signal Propagation delay Line (BPL) delay line, we have shown the ability to learn 11 of the 14 linearly separable functions with an accuracy of greater than 85%. Connecting these two delay line models with other systems also demonstrates the modular nature of the delay line system. A MDL can precisely store values at the expense of manual signaling and our BPL can do the same for smaller length memories without the need of manual control signaling. Our MDL

also demonstrates a model capable of storing values with less than 0.01% error.

8.1 Contributions

This work has made the following contributions to the field, ordered by appearance in this work:

- a new type of memory implemented in a Chemical Reaction Network, Manual copy Delay Line, in Chapter 3;
- a new type of memory implemented in a Chemical Reaction Network, Backwards signal Propagation delay Line, in Chapter 3;
- the first chemical model capable of learning binary time series with the combination of the delay line and Asymmetric Signal Perceptron, in Chapter 3;
- a framework capable of simulating ant trail problems with user customizable parameters on Artificial Neural Network and Genetic Algorithm parameters, in Chapter 4;
- a web based application capable of navigating, filtering, and viewing data from simulations on ant trail problems with ease, in Chapter 4;
- a novel architecture with a single hidden perceptron for solving the ant trail problems with the addition of a delay line as a memory, in Chapter 5;
- an investigation into the minimal length of delay line length of four to solve the artificial ant problem in a non-Chemical Reaction Network configuration, in Chapter 5;

- the first Chemical Reaction Network implementation to solve the artificial ant problem, in Chapter 6;
- the first Artificial Neural Network implemented in a Chemical Reaction Network to compare system level functionality against other work, in Chapter 6;
- evidence that a single hidden neuron when paired with a Manual copy Delay Line of length 4 is capable of solving 47% of the John Muir trail, 44% of the full Santa Fe trail, 100% of the easy Santa Fe trail segment, 84% of the medium Santa Fe trail segment, and 47% of the hard Santa Fe trail segment, in Chapter 6;
- proof that a Artificial Neural Network without a hidden neuron when paired with a Manual copy Delay Line of length 4 is capable of solving 47% of the John Muir trail, 57% of the full Santa Fe trail, 45% of the easy Santa Fe trail segment, 84% of the medium Santa Fe trail segment, and 47% of the hard Santa Fe trail segment, in Chapter 6.

8.2 Future Work

As discussed in Chapter 6, the GA used on CRN simulations was based off the GA optimization performed in the non-CRN environment. Further optimization of the the GA once the system is implemented as a CRN may yield better results for the overall system. The downside with such optimization is the long time to run simulations as discussed in Chapter 6. As the speed to run the simulations decreases, such evaluations may be more feasible in the future.

Banda has recently introduced a new type of delay line known as the parallel-accessible delay line [75]. This delay line behaves similar to the MDL, but adds

a clock signal that does not require the manual signaling of the MDL. The use of this delay line may in some instances reduce the complexity of the CRN reaction series.

We also believe that an implementation as a wet chemistry is a good next step. The current technology may limit a full implementation, but starting with a smaller system like the MDL, BPL, or the preceptron models presented by Banda *et al.* [7] [64] [13], would provide a first step towards a full wet chemistry realization.

Further work could also look at other areas of using the delay line to build more complex systems for interesting applications. As an example, one could pair the delay line with a chemical system acting as XOR to build a Linear Feedback Shift Register (LFSR). Development of a LFSR would lead to the ability to perform random number generation [82] in a CRN.

References

[1] B. De Lacy Costello and A. Adamatzky, "On multitasking in parallel chemical processors: experimental findings," *International Journal of Bifurcation and Chaos*, vol. 13, no. 02, pp. 521–533, 2003.

[2] N. Kanopoulos and J. J. Hallenbeck, "A first-in, first-out memory for signal processing applications," *Circuits and Systems, IEEE Transactions on*, vol. 33, no. 5, pp. 556–558, 1986.

[3] A. Arkin and J. Ross, "Computational functions in biochemical reaction networks," *Biophysical Journal*, vol. 67, no. 2, pp. 560–578, 1994.

[4] T. Meyer and C. Tschudin, "Flow management in packet networks through interacting queues and law-of-mass-action scheduling," Technical Report CS-2011-001, University of Basel, Tech. Rep., 2011.

[5] N. Matsumaru *et al.*, "Chemical organization theory as a theoretical base for chemical computing," in *Unconventional Computing 2005: From Cellular Automata to Wetware*, C. Teuscher and A. Adamatzky, Eds. Frome, United Kingdom: Luniver Press, 2005, pp. 71–82.

[6] E. Katz, *Biomolecular Information Processing: From Logic Systems to Smart Sensors and Actuators*, 1st ed. Weinheim, Germany: Wiley-VCH, 2012.

[7] P. Banda *et al.*, "Online learning in a chemical perceptron," *Artificial life*, vol. 19, no. 2, pp. 195–219, 2013.

[8] E. Katz, *Molecular and Supramolecular Information Processing: From Molecular Switches to Logic Systems*, 1st ed. Weinheim, Germany: Wiley-VCH, 2012.

[9] H. Jiang *et al.*, "Discrete-time signal processing with DNA," *ACS Synthetic Biology*, vol. 2, no. 5, pp. 245–254, 2013.

[10] M. N. Stojanovic and D. Stefanovic, "A deoxyribozyme-based molecular automaton," *Nature Biotechnology*, vol. 21, no. 9, pp. 1069–1074, 2003.

[11] D. Faulhammer *et al.*, "Molecular computation: RNA solutions to chess problems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 97, no. 4, pp. 1385–1389, 2000.

[12] D. Jefferson *et al.*, "The genesys system: Evolution as a theme in artificial life," in *Proceedings of Second Conference on Artificial Life*, C. G. Langton *et al.*, Eds. Redwood City, California, USA: Addison-Wesley Publishing Company, 1992, pp. 549–578.

[13] P. Banda and C. Teuscher, "Learning Two-Input linear and nonlinear analog functions with a simple chemical system," in *Unconventional Computation and Natural Computation*, O. H. Ibarra *et al.*, Eds., vol. 853. Sennweid, Switzerland: Springer International Publishing, 2014, pp. 14–26.

[14] D. Blount *et al.*, "Feedforward chemical neural network: A compartmentalized chemical system that learns XOR," unpublished.

[15] D. H. Scheidt, "Intelligent agent-based control," *John Hopkins APL Technical Digest*, vol. 23, no. 4, 2002.

[16] G. W. Neat *et al.*, “A hybrid adaptive control approach for drug delivery systems,” in *Engineering in Medicine and Biology Society, 1988. Proceedings of the Annual International Conference of the IEEE.* Piscataway, New Jersey, USA: IEEE, 1988, pp. 1305–1306 vol.3.

[17] J. Halámek *et al.*, “Multi-enzyme logic network architectures for assessing injuries: digital processing of biomarkers,” *Molecular BioSystems*, vol. 6, no. 12, pp. 2554–2560, 2010.

[18] M. F. Abbod *et al.*, “Survey on the use of smart and adaptive engineering systems in medicine,” *Artificial Intelligence in Medicine*, vol. 26, no. 3, pp. 179–209, Nov. 2002.

[19] J. Wang and E. Katz, “Digital biosensors with built-in logic for biomedical applications-biosensors based on a biocomputing concept,” *Analytical and Bioanalytical Chemistry*, vol. 398, no. 4, pp. 1591–1603, 2010.

[20] M. Zhou *et al.*, “A self-powered “sense-act-treat” system that is based on a biofuel cell and controlled by boolean logic,” *Angewandte Chemie*, 2012.

[21] S. Mailloux *et al.*, “A model system for targeted drug release triggered by biomolecular signals logically processed through enzyme logic networks,” *The Analyst*, vol. 139, no. 5, pp. 982–986, 2014.

[22] ——, “Enzymatic filter for improved separation of output signals in enzyme logic systems towards ’sense and treat’medicine,” *Biomaterials Science*, vol. 2, no. 2, pp. 184–191, 2014.

[23] J. Alvarez *et al.*, "Development of a multiplex PCR technique for detection and epidemiological typing of salmonella in human clinical samples," *Journal of Clinical Microbiology*, vol. 42, no. 4, pp. 1734–1738, 2004.

[24] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems).* Cambridge, Massachusetts, USA: A Bradford Book, 1992.

[25] J. Doucette and M. I. Heywood, "Novelty-Based fitness: An evaluation under the Santa Fe trail," in *Genetic Programming.* Heidelberg, Germany: Springer Berlin Heidelberg, 2010, pp. 50–61.

[26] S. Christensen and F. Oppacher, "Solving the artificial ant on the Santa Fe trail problem in 20,696 fitness evaluations," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation.* New York, New York, USA: ACM, 2007, pp. 1574–1579.

[27] P. Gargesa, "Reward-driven training of random boolean network reservoirs for Model-Free environments," Master's thesis, Portland State University, 2013.

[28] D. Wilson and D. Kaur, "How Santa Fe ants evolve," *ArXiv e-prints*, 2013. [Online]. Available: http://arxiv.org/abs/1312.1858

[29] D. S. Chivilikhin *et al.*, "Solving five instances of the artificial ant problem with ant colony optimization," in *7th IFAC Conference on Manufacturing Modelling, Management, and Control*, N. Bakhtadze *et al.*, Eds. St. Petersburg, Russia: International Federation of Automatic Control, 2013, pp. 1043–1048.

[30] M. R. Karim and C. Ryan, "Sensitive ants are sensible ants," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation.* New York, New York, USA: ACM, 2012, pp. 775–782.

[31] A. Silva *et al.*, "Evolving controllers for autonomous agents using genetically programmed networks," in *Genetic Programming.* Heidelberg, Germany: Springer Berlin Heidelberg, 1999, pp. 255–269.

[32] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[33] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[34] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, expanded ed. Cambridge, Massachusetts, USA: The MIT Press, 1987.

[35] R. Rojas, *Neural Networks: A Systematic Introduction.* Heidelberg, Germany: Springer Berlin Heidelberg, 1996.

[36] D. E. Rumelhart *et al.*, "Learning representations by back-propagating errors," in *Cognitive Modeling*, T. Polk and C. Seifert, Eds. Cambridge, Massachusetts, USA: A Bradford Book, 1988, pp. 213–220.

[37] J. H. Holland, "Outline for a logical theory of adaptive systems," *Journal of the ACM*, vol. 9, no. 3, pp. 297–314, 1962.

[38] ——, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Cambridge, Massachusetts, USA: A Bradford Book, 1975.

[39] M. Mitchell, *An Introduction to Genetic Algorithms*, 3rd ed. Cambridge, Massachusetts, USA: MIT Press, 1998.

[40] P. J. Fleming and R. C. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control engineering practice*, vol. 10, no. 11, pp. 1223–1241, 2002.

[41] S. K. Pal *et al.*, "Genetic algorithms for optimal image enhancement," *Pattern recognition letters*, vol. 15, no. 3, pp. 261–271, 1994.

[42] F. Menczer and R. K. Belew, "Evolving sensors in environments of controlled complexity," in *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, Eds. Cambridge, Massachusetts, USA: MIT Press, 1994, pp. 210–221.

[43] T. Baeck *et al.*, *Evolutionary Computation 1: Basic Algorithms and Operators.* Institute of Physics Publishing, 2000, vol. Bristol, United Kingdom.

[44] A. Brindle, "Genetic algorithms for function optimization," Ph.D. dissertation, University of Alberta, 1981.

[45] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Francisco, California, USA: Morgan Kaufmann, 1991, pp. 69–93.

[46] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms.* Hillsdale, New Jersy , USA: L. Erlbaum Associates Inc., 1985, pp. 101–111.

[47] H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie.* Basel, Switzerland: Birkh̀auser, 1976.

[48] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[49] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975.

[50] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing.* Norwell, Massachusetts, USA: Kluwer Academic Publishers, 1987.

[51] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 2–9.

[52] P. J. Angeline and K. E. Kinnear, *Advances in Genetic Programming.* Cambridge, Massachusetts, USA: Bradford Books, 1996.

[53] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the Traveling-Salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[54] J. Moles *et al.*, "Delay line as a chemical reaction network," *Parallel Processing Letters*, 2014, to be published.

[55] P. Dittrich *et al.*, "Artificial Chemistries-A review," *Artificial life*, vol. 7, no. 3, pp. 225–275, 2001.

[56] F. Horn and R. Jackson, "General mass action kinetics," *Archive for rational mechanics and analysis*, vol. 47, no. 2, pp. 81–116, 1972.

[57] P. Érdi and J. Tóth, *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*. Manchester, England: Manchester University Press, 1989.

[58] V. Henri, *Lois Générales de l'Action des Diastases*. Paris, France: Librairie Scientifique A. Hermann, 1903.

[59] L. Michaelis and M. L. Menten, "Die Kinetik der Invertinwirkung," *Biochemische Zeitschrift*, vol. 49, pp. 333–369, 1913.

[60] V. Leskovac, *Comprehensive Enzyme Kinetics*, V. Leskovac, Ed. Dordrecht, the Netherlands: Kluwer Academic / Plenum Publishers, 2003.

[61] I. R. Epstein and J. A. Pojman, *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. Oxford, England: Oxford University Press, 1998.

[62] R. A. Copeland, *Enzymes: A Practical Introduction to Structure, Mechanism, and Data Analysis*. Hoboken, New Jersey, USA: Wiley, 2004.

[63] P. Banda *et al.*, "COEL: A web-based chemistry simulation framework," in *CoSMoS 2014: Proceedings of the 7th Workshop on Complex Systems Modelling and Simulation*, S. Stepney and P. Andrews, Eds. Frome, United Kingdom: Luniver Press, 2014, pp. 35–60.

[64] ——, "Training an asymmetric signal perceptron through reinforcement in an artificial chemistry," *Journal of the Royal Society, Interface / the Royal Society*, vol. 11, no. 93, p. 20131100, 2014.

[65] M. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers. Institute of Electrical and Electronics Engineers*, vol. C-21, no. 9, pp. 948–960, 1972.

[66] T. Schaul *et al.*, "PyBrain," *Journal of Machine Learning Research: JMLR*, vol. 11, pp. 743–746, 2010.

[67] F.-A. Fortin *et al.*, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research: JMLR*, vol. 13, no. 1, pp. 2171–2175, 2012.

[68] Y. Hold-Geoffroy *et al.*, "Once you SCOOP, no need to fork," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment.* New York, NY, USA: ACM, 2014, pp. 60:1–60:8.

[69] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database," 1996, accessed: 2014-11-5. [Online]. Available: http://www.postgresql.org/

[70] A. Ronacher, "Flask (a Python microframework)," 2010, accessed: 2014-11-5. [Online]. Available: http://flask.pocoo.org/

[71] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[72] Plotly, "Plotly," 2012, accessed: 2014-11-2. [Online]. Available: https://plot.ly/

[73] K. A. De Jong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Heidelberg, Germany: Springer Berlin Heidelberg, 1991, pp. 38–47.

[74] DigitalOcean Inc., "SSD cloud server, VPS server, simple cloud hosting," accessed: 2014-11-7. [Online]. Available: https://www.digitalocean.com/

[75] P. Banda and C. Teuscher, "An analog chemical circuit with Parallel-Accessible delay line for learning temporal tasks," in *ALIFE 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, H. Sayama *et al.*, Eds. Cambridge, Massachusetts, USA: MIT Press, 2014, pp. 482–490.

[76] K. L. Kompa and R. D. Levine, "A molecular logic gate," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 2, pp. 410–414, 2001.

[77] D. Y. Zhang and G. Seelig, "Dynamic DNA nanotechnology using strand-displacement reactions," *Nature chemistry*, vol. 3, no. 2, pp. 103–113, 2011.

[78] O. Semenov *et al.*, "The effects of multivalency and kinetics in nanoscale search by molecular spiders," in *Evolution, Complexity and Artificial Life*, S. Cagnoni *et al.*, Eds. Springer Berlin Heidelberg, 2014, pp. 161–175.

[79] M. N. Stojanovic *et al.*, "Homogeneous assays based on deoxyribozyme catalysis," *Nucleic acids research*, vol. 28, no. 15, pp. 2915–2918, 2000.

[80] J. Liu *et al.*, "Functional nucleic acid sensors," *Chemical Reviews*, vol. 109, no. 5, pp. 1948–1998, 2009.

[81] L. Qian *et al.*, "Neural network computation with DNA strand displacement cascades," *Nature*, vol. 475, no. 7356, pp. 368–372, 2011.

[82] P. P. Chu and R. E. Jones, "Design techniques of FPGA based random number generator," in *Military and Aerospace Applications of Programmable Devices and Technologies Conference.* klabs.org, 1999.