A MODEL FOR THE EVOLUTION OF NUCLEOTIDE POLYMERASE DIRECTIONALITY

by

Joshua Ballanco

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Joshua Ballanco, Candidate

ADVISORY COMMITTEE

Marc Mansfield, Chairman Date

A. K. Ganguly Date

Knut Stamnes Date

Nicholas Murgolo Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2009

A MODEL FOR THE EVOLUTION OF NUCLEOTIDE POLYMERASE DIRECTIONALITY

ABSTRACT

It is my thesis that life can be modeled as a thermodynamic system. Specifically, it is my goal to use this approach to understanding life in order to investigate evolution as an optimization problem. This justification for this approach is derived from the observation that life is, at its most fundamental level, an elaborate collection of chemical reactions. Individually, each of these reactions is governed by the laws of thermodynamics, and thus far there is no known upper limit to the scale at which thermodynamic principles may be applied.

In this work I present a model systems which is inspired by this hypothesis. In this model, I will look at the directionality of nucleotide polymerases, all of which synthesize new nucleotide polymers in a 5' to 3' direction. This phenomenon could be the consequence of a very early founder effect. On the other hand, it could be that this directionality evolved due to an inherent advantage. I would propose that the process of nucleic acid polymerization provides a clue as to what that advantage might have been. The model I have constructed demonstrates this, and the results provide an insight into the forces that might have driven early biological evolution.

Author: Joshua Ballanco

Advisor: Marc Mansfield

Date: Sept. 18, 2009

Department: Chemistry, Chemical Biology, and Biomedical Engineering

Degree: Doctor of Philosophy

Table of Contents

List of Figures

Chapter 1

Introduction

Evolution has been, since the time of Darwin, a science mostly concerned with the past. The principles of evolution have been used to explain observations of ancient creatures and to predict what new evidence from the past should eventually turn up. One example of this sort of prediction is the existence of transitional forms. Evolution through the gradual acquisition of altered traits would predict the existence of ancient animals with a blending of traits from closely related branches in the tree of life. Indeed many such forms have been discovered, validating this prediction.

When it comes to predicting the past, evolution has been wildly successful. Where the study of Evolution has been thus far lacking is in its ability to predict the future. That is, looking at collected fossils and current natural evidence, the Theory of Evolution gives us the ability to identify which selective pressures acted on past populations. What the Theory of Evolution cannot do, at present, is predict which environmental or other influences will act as selective pressures going forward. At best, we can make educated guesses based on past evidence, but we lack even the basic ability to assign a concrete measure of confidence in such predictions.

This situation is akin to a meteorologist being able to explain why it rained yesterday but completely unable to predict tomorrow's weather. It is important, though, to understand that this is not an inherent failing of evolution, but rather a sign of a young science with much work left to do and many discoveries yet to be made. In physics, Newton's laws were successful at explaining the action of an apple falling from a tree for many centuries before they were developed to the point they could inform us how to send a man to the moon. What's remarkable about this development, from falling apples to moon landings, is that it did not require any fundamental additions to Newton's laws. (Certainly General Relativity has fundamentally altered Newton's laws, but it is not, strictly speaking, an alteration required to get to the moon.) Rather, physics was able to make this progress merely through improved tools, improved instruments, and most importantly, improved means of applying the fundamental laws to a problem.

Thus, in order to advance the field of evolution, we should strive for more and better data, but also new and better ways of analyzing and testing that data. Before looking at how this might be achieved, let us look at what the implications of better predictive power in evolution might be. A rather straight forward implication would be the ability to predict the occurrence and course of epidemic or even pandemic diseases. Such diseases are biological organisms subject to Darwinian evolution and, often times, humans are the niche for which they are adapting. We very frequently alter their niche, introducing new selective pressures, in the ways that we treat disease with medicines, quarantine, or myriad other techniques.

The implications can be more far reaching then they might first seem. Partially, this is due to the fact that evolution, and the principles of Darwinian Evolution, apply to a much more diverse range of situations than just the origin of animal and plant species. For example, cancer is an evolutionary process. With each round of chemotherapy or radiation treatment, those cancer cells that have adaptations that increase their resistance to treatment will be more likely to survive. These cells will, thus, seed what will almost inevitably form as a reemergent, more difficult to treat, tumor. Therefore, understanding the dynamics of evolution and how to predict the future course of evolution would improve our ability to design effective treatments for cancer.

Even non-biological processes are driven by evolution and obey many of the same laws that Darwin first laid out 150 years ago. Both languages and economies undergo evolution, driven by the same math as cancer or the origin of species. At this point it is useful to make a distinction between biological and non-biological evolution. The reason for doing so lies in the approaches that can be taken to investigate each type of evolutionary system. We know a great deal more about biological organisms than that they merely evolve. The past 50 years has resulted in an explosion of understanding of the chemistry of life and the operation of the molecular systems which compose cells. On the other hand, systems such as language and economy have, as their atomic components, humans and the human mind. While we understand much about the human mind, our grasp of its elementary functioning still pales in comparison to recent advances in biochemistry and molecular biology.

For this reason, while non-biological systems can be studied using an outside-in approach just as easily as biological systems, biological evolving systems present to us a unique opportunity to attempt to understand the mechanisms of evolution from the inside. Specifically, with biology we can explore the internal feedback mechanism which drives evolution: the “Central Dogma” of biology. This is the pathway by which information flows from an organisms nucleic acids, where it is stored, to the organisms proteins, where the information drives fundamental biochemical processes. These biochemical processes are what is eventually selected for in the process of natural selection, determining what information gets propagated, but these biochemical processes are also what carries out the propagation of that information.

To understand why it is so difficult to make predictions about the future dynamics of evolutionary systems, it helps to the details of how such systems works. Darwin’s essential observations can be summed up in two important concepts: reproductive success and descent with modification. The concept of reproductive success states that those organisms which reproduce in the greatest number and at the greatest rate will have their genes increase in frequency in a population over time. Since it is these genes which determine, to some extent or another, how successful organisms will be at reproducing, reproductive success serves as a filter for so-called "best fit" genes. Such a filter would, ultimately, result in a highly refined population of only those organisms which represent the best possible reproducers. However, such a system is static, and we know from observation that evolving populations are dynamic.

The dynamism of evolving populations comes about from descent with modification. In every biological organism, and indeed in every reproductive unit which experiences Darwinian evolution, the process of reproduction is not carried out with perfect fidelity. Rather, in each generation new variants arise. These variants can be generated through a number of mechanisms such as the recombination of distinct genetic elements or the blending of traits from multiple parents. If we are to understand biological evolution at its most basic, however, the most important mechanism for the introduction of variation in early organisms is mutation: errors in the fundamental process of duplicating genetic information.

Restating the problem of making predictions about evolutionary processes, then, what we must be concerned with are which filters will operate on a population through the process of reproductive success and how the processes that introduce variation will be affected by and at the same time effect these reproductive success filters. This conceptual model is familiar from the science of thermodynamics. In thermodynamics a system particles is subject to certain forces. The particles are also imparted with certain momenta. The trajectories described by these momenta determine which forces and with what magnitude the particles will experience as time progresses. At the same time, the forces each particle is subject to will alter the particle's trajectory. So, in a thermodynamics inspired model of evolution, we can think of the selective filters of reproductive success as forces and the introduction of new variants as momenta. In this work I will explore the applicability of this conceptual model by looking at the specific case of nucleotide polymerases. This exploration takes the form of a model of polymerase evolution.

Nucleic Acid Polymerization

The two classes of biologically important nucleic acids are Ribonucleic Acid (RNA) and Deoxyribonucleic Acid. These two classes of molecules are very similar, differing only by the presence or absence, respectively, of a 2' hydroxyl group on the ribose sugar of their individual monomers. Both are formed primarily by a process of dehydration synthesis catalyzed by a class of enzymes known as nucleotide polymerases. The dehydration reaction takes place between one of the hydroxyl groups on the α phosphate of a nucleotide triphosphate and the 3' hydroxyl group of the terminal monomer on the growing nucleic acid chain.

What is phenomenal about this process is that, first, it occurs in all biological organisms and, second, that it always occurs with the same directionality. To understand why the consistency of directionality is notable, it helps to understand the catalytic process that occurs at the active center of nucleotide polymerases. All known nucleotide polymerases share a common chemical mechanism. In this mechanism, two divalent metal cations, coordinated by a number of acidic amino acids, facilitate the transfer of an electron pair from the free 3' hydroxyl group of one nucleotide to the α phosphate, which is attached to the 5' hydroxyl of the other nucleotide.

In this mechanism, the catalytically active cations and acidic amino acids are unbiased as to which nucleotide presents the free 3' hydroxyl and which presents the 5'-linked phosphate group. In other words, if one were to imagine a nucleotide polymerase that proceeded in the reverse direction of all currently known nucleotide polymerases, the only aspect of the naturally occurring enzymes that would need to be modified are those portions which attach to the growing nucleotide polymer or the incoming nucleotide, all of which are distinct from the catalytic center. Yet, the fact that all biological organisms polymerize both DNA and RNA with the same directionality implies that this aspect of nucleotide polymerization was decided extremely early on in the genesis of life on earth, most likely even before the biological distinction between DNA and RNA functionality arose.

If the active site mechanism of nucleotide polymerases cannot explain the apparent bias of all life toward one polymerization directionality over the other, then what alternative explanations might exist? Two possibilities are that the unique directionality is the consequence of a founder effect or that there is an evolutionary advantage to the selected directionality. A founder effect is the result of when a small subset of a larger population is evolutionarily isolated from the original population and goes on to give rise to a new population. This new population would be expected to contain an oversampling of the genetics of the founding population as compared to the gene frequencies of the original population. In the case of nucleotide polymerization, this would imply that at some point early on in the development of life on earth the population contained both forms of nucleotide polymerases, those that polymerize by extension $3' \rightarrow 5'$ and $5' \rightarrow 3'$ (as all life today). Then, at some later point, a subgroup of this population that contained only $5' \rightarrow 3'$ polymerases was isolated and subsequently gave rise to all life on earth today.

Unfortunately, the chances of finding any evidence directly supporting the hypothesis of a founder effect determining polymerase directionality are essentially nil. Short of finding some remnant modern population with reversed polymerases, the only evidence of an ancient $3' \rightarrow 5'$ polymerase would be the enzymes or other early replicator molecules themselves, and individual molecules do not fossilize. Therefore, to shed more light on which of these two competing hypotheses explains the current state of nucleotide polymerases in biology, it will be necessary to identify what possible evolutionary advantage could have been imparted on an organism by having a $5' \rightarrow 3'$ polymerase instead of the reverse. In order to do this, the forces and momenta that would influence the evolution of a nucleotide polymerase in the simplest of replicating proto-organisms should be considered, in keeping with a thermodynamic approach to evolution.

Because nucleotide polymerases are responsible for replicating the genetic information of an organism, and therefore have a direct influence on both the rate at which an organism can reproduce as well as the fidelity with which genetic information is conveyed from one generation to the next, this analysis is relatively straight forward. The model presented here will involve simplified model organisms designed to represent the earliest forms of self-replicating life. For the model, it is assumed these organisms must have arisen in an environment rich with nutrients and an excess of available energy. This means that the rate of genome duplication would serve as the limiting factor on reproductive rate. This is one force that will act on the evolution of the model organisms.

In addition to reproductive rate, we must also take into account the possibility of a lower acceptable limit on genetic transmission fidelity. At the limit, a polymerase which has zero fidelity in reproducing genetic information cannot really be said to be evolving so much as randomly assembling chemicals. That is, while the dynamic nature of evolution requires the introduction of some information entropy during reproduction, without at least some of the original message being persisted across generations selective pressures cannot operate. The precise value of this lower limit will depend on a number of factors. For the model constructed here, a more pragmatic approach has been adopted. It has been observed empirically that, averaged over a large sampling of proteins and organisms, at least 76% fidelity is required for continued function. Thus, the second force which will operate on the model organisms is that of minimum acceptable polymerase fidelity.

As for the momentum which will guide the model organisms through the evolutionary landscape, only mutation rate will be considered. Since polymerase directionality would have been fixed extremely early on in the origin of life, it is reasonable to assume that the recombination of genetic elements would contribute only a negligible amount of variation to the model organisms. Additionally, the simplest of nucleotide polymerases observed in nature are the products of single genes. Even if there was recombination of individual genetic elements, the exclusive focus of this model is the polymerase. Whether a product polymerase gene is transferred intact to the offspring of one organism or transferred horizontally will not affect the conclusions that can be made. This is a result of the first assumption made that the model organisms have an excess of all resources aside from the polymerase. That is, whatever organism a polymerase may end up in, it will always be the single determining factor for both reproductive rate and fidelity.

Chapter 2

Design of the Polymerase Evolution Model

This chapter provides a detailed description of the design of the model system used to investigate the question of the evolution of nucleotide polymerase directionality. The goal of this model is to have organisms containing either a $5' \rightarrow 3'$ or a $3' \rightarrow 5'$ nucleic acid polymerase compete with each other in order to see which of the strategies, if either, are evolutionarily dominant. In the model, evolutionary pressure is applied by the rate at which organisms can reproduce and the fidelity with which they duplicate their genomes. Evolutionary momentum is introduced through simple genetic mutation. The model is also designed such that the influence of temperature on the outcome of this competition can be investigated.

Overview

In simplest terms, this model consists of a number of individual organisms in competition with each other. The model can be largely divided up into, and thought about most clearly as, four interacting pieces. These are the environment, the organisms, the genomes, and the polymerases. The relationship between these pieces begins with the environment. Each run of the simulation involves exactly one environment. An environment may contain many organisms, up to a predefined carrying capacity, and at the beginning of a simulation run the environment is pre-populated organisms according to a set rules.

Each organism contains one genome. At the start of an organisms "life", it uses its genome to create a polymerase. The polymerase replicates either $5' \rightarrow 3'$ or $3' \rightarrow 5'$ and may not change direction. The polymerase then replicates the genome at a predetermined rate and with a random chance of introducing errors (mutations). When the polymerase is finished duplicating the genome, the organism will attempt to divide by binary fission using the newly created genome. If it is successful in doing so, then the new organism is added to the environment and begins a fresh life cycle while the original organism resets itself and also begins a fresh cycle.

A number of generalizations and assumptions have been made in order to render the problem being investigated tractable, but a consistent effort has been made to remain as true to life as possible. The motivation behind this is that, while the exact values generated by this model may not be precisely those that would be observed in the laboratory, the trends observed should hold true when transferred to the bench. For each piece decisions have been made regarding which aspects of the component are explored in depth, and which aspects are neglected, with an eye to the larger goal of investigating the role of polymerase directionality. Following is a detailed look at each component in turn.

Environment

At the start of a simulation run, the environment for that run is initialized with a starting population of organisms. To create this starting population, a set of organisms with their genome length, polymerase rate, and polymerase directionality specified is divided according each organism's designated frequency in the starting population. The starting population size can be any number up to the specified maximum population of the environment.

A key defining feature of this model is that the environment is constrained in some ways, but not in others. Specifically, the number of organisms that can simultaneously co-exist has a hard numerical limit; a carrying capacity. On the other hand, the amount of available energy, the quantity of activated nucleotide triphosphates, and the other raw materials required for forming a cell are all considered unlimited. In reality, the carrying capacity is a generalization that could correspond to physical space limits, but it could just as easily correspond to the aggregate limitations on the other resources not explicitly modeled.

The choice of limitation based on carrying capacity was driven by two factors. The first relates to the fact that selection during an exponential growth phase could potentially operate differently than during stationary phase growth. By placing an upper limit on the size of a population, it is possible to investigate selection during both growth phases. Second, while attempting to explicitly model all of the various resource and space constraints that might ultimately limit growth might yield a more complete model, it would also significantly increase the complexity of the model without adding much insight into the question at hand. Finally, numerically or density limited growth is a common observation across a wide range of living creatures from single cells to large populations of complex animals.

In order to model density dependent growth inhibition as the number of organisms approachs the carrying capacity, a random death probability is introduced to the environment. In keeping with observations that the pressure of density dependant inhibition is greater as the population of an environment approaches the carrying capacity, the death probability is calculated with an inverse function of the remaining capacity

$$P_{death} = \frac{1}{(N - n) + 1}$$

where N is the carrying capacity, n is the number of organisms currently in the environment, and 1 is added so that the probability of at least one organism being culled from the environment when the carrying capacity is reached is $P_{death} = 1$.

The model iterates its environment in a stepwise fashion. During each step, each of the organisms contained within the environment is allowed to carry out one time-step of its life cycle, followed by a population culling. Culling is carried out by calculating the death probability as above, and then randomly applying that probability to the environment to decide if an organism should be removed. If the decision is made to remove an organism, one is chosen from the environment randomly, removed, and the death probability is recalculated and reapplied. This process will repeat until the decision is made not to remove an organism. In this way, the number of organisms removed at each time step follows a Poisson distribution with an expectation value of P_{death}.

Organism

Organisms are created and added to the environment either at the start of a simulation run, as part of the starting population, or during the simulation run as the result of binary fission of an existing organism. If the organism is created at the start of the simulation then its genome and polymerase properties are determined by the values used to seed the population. If it is created during a simulation run, then its properties are derived from those of its parent, with the possibility of introduced variation. This variation is embodied by, and in the model determined by, the genome contained within the organism. Each organism contains exactly one genome and one polymerase. Having only one polymerase is a simplification, but it is justified by considering this one polymerase as an exemplar of the various polymerase enzymes that would be found in a real organism.

Each organism is modeled as a state machine. The two states in which they can exist are: *Polymerizing* or *Duplicating*. At creation, each organism starts in the

Polymerizing state. When an organism is in the *Polymerizing* state, each simulation time-step is used to allow the organism's polymerase to add nucleotides to the genome copy being constructed. At the end of each time-step, the polymerase is queried as to whether or not it is finished with constructing the nascent genome. If it is, then the organism shifts to the *Duplicating* state. Otherwise, it remains in the *Polymerizing* state.

When an organism reaches the *Duplicating* state, the first task is to determine if the genome copy constructed by the polymerase is viable. This determination is made by comparing the fraction of errors made by the polymerase during polymerization against the empirically observed upper tolerance of 76%. If the genome copy is not viable, then it is discarded and the organism returns to the *Polymerizing* state to create a new genome copy. That is, no allowance is made for damage repair. This is yet another simplifying assumption, but considering that the model aims to simulate the veriest early forms of life, it is doubtful that complex *post facto* error correcting mechanisms would have existed (or that they would have contributed significantly to the evolution of such early organisms).

If the genome is viable, the next determination that must be made is whether or not there is available capacity in the environment for a new organism. If the number of organisms in the environment is not at the environment's carrying capacity when the environment tells the organism in the *Duplicating* state to step, then a new organism containing the newly synthesized genome is created and inserted into the environment. In this case, the parent organism will return to the *Polymerizing* state to begin construction of another new genome. If, instead, the environment is at capacity when an organism, with a viable genome product, in the *Duplicating* state is told by the environment to proceed one time-step, the organism will remain unchanged, in the *Duplicating* state. It will next attempt to add a new organism with the synthesized genome during the next environment step, but nothing will change until then.

Genome

Genomes are created before the organism in which they are contained. In the case of a running simulation, the new genome is created by the parent organism before binary fission. For the starting population of organisms used to seed a simulation, the genomes are created ahead of time with predefined properties. The model genomes perform three important functions for the simulation: they generate polymerases with properties defined by the genome, they track the progress of the organism's polymerase during polymerization, and they return a copy of themselves once polymerization has completed.

Each genome codes for either a $5' \rightarrow 3'$ polymerase or a $3' \rightarrow 5'$ polymerase, and this will not change through subsequent generations. There is the possibility that this is not perfectly reflective of how early organisms may have functioned, as it is conceivable that, through acquired mutations, a genome which coded for a $3' \rightarrow 5'$ polymerase may evolve in subsequent generations into a genome that codes for a $5' \rightarrow 3'$. Were this to occur, it would either have to be through a gradual mechanism or a small mutation affecting global properties of the polymerase. A gradual mechanism would be one where the forward or reverse speed of the polymerase is reduced to nearly zero, and then eventually the directionality flipped. It is safe to discount this possibility in the model, as such a transition would require the survival of an organism with an especially slow polymerase. As a consequence of the starting assumption that polymerase speed is the primary determining factor in the speed of replication, such organisms would be quickly out-competed in the simulation.

The alternative mechanism of a small mutation leading to a global change can also be neglected for two reasons. First, if such a mutation were possible, the fidelity of communicating polymerase directionality from one generation to the next would be diminished. This would imply that selecting against a $3' \rightarrow 5'$ polymerase would effectively be impossible, as such polymerases could appear at almost any time. This would also narrow the window of possible founder populations were the alternate founder effect hypothesis considered. In the case that polymerase directionality were trivially reversible, the hypothetical founder population would not only have to be homogenous in their polymerase directionality, but they would also have to have a unique restriction on the directionality reversing randomly. This also highlights the second reason that the possibility of such a mutation need not be considered: if such mutations were possible, it would be expected that they could be observed today. Certainly, it is possible that such a mutation might have existed in the past but have subsequently been locked out of the population of current polymerase sequences, but this seems unlikely. Nucleotide polymerases are wildly variable in their sequence and structure throughout all forms of life, and yet none exhibit reversed directionality.

When a genome is asked by its organism to create a polymerase, it will imbue the polymerase with a fixed polymerase rate in addition to a fixed directionality. The precise value the polymerase speed will take is determined when the genome is created from its parent (or from predefined values at the beginning of a simulation). The variance from its parent genome's polymerase speed is derived from the amount of acquired mutations as described below. Once the polymerase has been generated and polymerization begun, the polymerase will inform the genome during each simulation time-step how many nucleotides have been added to the copy being created, and how many of those are erroneous base pairs. The genome keeps track of this information for later reporting and calculations.

Finally, a genome can be queried as to the viability of the copy constructed and asked to return that copy. The determination of viability is simply calculated by taking the percent errors ($\frac{E}{L}*100$) where E is the number of errors and L is the length of the full genome). If this percentage is above the threshold, set at 34% based on empirical observations, then the genome is deemed inviable. Given a viable copy was constructed, the genome can then return the copy but, in doing so, it must determine the properties of this new genome. As mentioned above, the directionality is inherited exactly. The polymerase speed coded for by the copy genome is determined in a two step calculation based on the number of errors (mutations) introduced. First, the variance from the parent genome's encoded polymerase rate is calculated by taking

$$\Delta = \frac{r_{max} - r_{min}}{2} * \frac{M}{M_{max}}$$

where r_{max} and r_{min} are the maximum and minimum possible rates, respectively, M is the number of errors made during replication, taken as a fraction of total genome length, and M_{max} is the maximum tolerated fraction of errors. The maximum and minimum rates were chosen as 1 and 10 nucleotides added per simulation time-step to reflect the roughly 10-fold range of empirically observed polymerase rates. The logic behind this metric is that more mutations will, on average, lead to greater alteration of enzyme function. Certainly there are individual mutations that could, on their own, have a large effect on polymerase rate, but averaged over many polymerases and many possible mutations, this general trend should hold.

Once the variance is determined, it must be applied to the parent genome's encoded polymerase rate in such a way that the child's genome does not code for an invalid polymerase. So, if the variance would generate a polymerase rate above the maximum, then it is subtracted from the parent rate, and vice versa. If the variance would not result in an invalid rate being either added or subtracted, then it is applied to the parent rate with an equal probability of either outcome.

Polymerase

The single most significant component of the model is the model nucleotide polymerase itself. The other three components described above serve primarily to ensure that selection and descent with modification will occur, but it is the polymerase which embodies the trait that is ultimately being selected for or against: the polymerase's directionality. As with the genomes, the directionality and rate of a polymerase does not change subsequent to its creation. These properties are determined by the genome from which the polymerase is generated. The polymerases only have one function during a simulation: to add nucleotides to a growing genome copy. How this occurs depends on the polymerase directionality and the temperature of the simulation.

When an organism in the *Polymerizing* state carries out one step of simulation time, it requests of its polymerase for nucleotides to be added to the genome being synthesized. The polymerase does this by running through a loop with a maximum number of iterations determined by its rate. So, for example, a polymerase with a rate of 7 can carry out its nucleotide addition loop a maximum of 7 times. During each loop, two determinations must be made. First, the polymerase must determine whether the nucleotide about to be added is activated, and second it must determine whether a properly base-paired nucleotide is being added, or whether the nucleotide being added will generate a replication error.

The reason that the polymerase must determine whether or not the nucleotide being added is activated is due to a consequence of chemistry. All known nucleotide polymerases in nature use, as their substrate, nucleotide triphosphates. They use the energy released by cleaving the bond between the α and β phosphates attached to their $5'$ carbon to drive the polymerase reaction forward. However, this bond can be cleaved through spontaneous hydrolysis by water. The equilibrium between active and inactive nucleotides can be described by a Boltzmann distribution:

$$K = e^{-\frac{\Delta G^0}{RT}}$$

where ΔG^0 is the standard free energy of the hydrolysis reaction. To simplify the simulation calculations, the entire $\frac{\Delta G^0}{RT}$ term is expressed as a generic simulation temperature $\frac{1}{t}$.

Where this process of inactivation becomes important is in how it will differentially affect a $5' \rightarrow 3'$ polymerase vs a $3' \rightarrow 5'$ polymerase. In the $5' \rightarrow 3'$ case, the one that is observed in all known forms of life, the triphosphate is attached to the incoming nucleotide about to be added to the chain. If this incoming nucleotide becomes inactive before the polymerase has a chance to join it to the growing nucleic acid polymer, then this represents a wasted addition step. That is, if a polymerase with a rate of 7 has one of its nucleotides become inactive before phosphodiester bond formation, then it will only add 6 nucleotides during this simulation time-step.

On the other hand, in the case of a hypothetical $3' \rightarrow 5'$ polymerase, the triphosphate group would be attached to the $5'$ carbon of the nascent nucleic acid polymer. If, during an addition step, this triphosphate is cleaved the polymerase will not be able to compensate by drawing in a new nucleotide from the pool of available nucleotides. Rather, the polymerization process will necessarily halt while the terminal end of the growing chain is reactivated. In the simulation, that process is modeled by terminating the polymerase's addition of new nucleotides when an inactivation event occurs. That is, if a $3' \rightarrow 5'$ polymerase with a rate of 7 experiences inactivation during the addition of the 3rd nucleotide in this time step, then only 2 new nucleotides will be added to the growing genome during this particular step.

Furthermore, whereas a $5' \rightarrow 3'$ polymerase will be constantly drawing in new activated nucleotides from the pool of all available nucleotides, a $3' \rightarrow 5'$ polymerase depends on the nascent chain remaining activated. The consequence of this is that the probability of a $5' \rightarrow 3'$ polymerase experiencing a deactivation event depends only on the free energy of the reaction and temperature, irrespective of polymerase rate. Alternatively, a $3' \rightarrow 5'$ polymerase runs a greater risk of the triphosphate on the nascent chain becoming deactivated the longer it waits before adding a new nucleotide. To model this, the probability of deactivation is multiplied by the difference between the rate of each $3' \rightarrow 5'$ polymerase and the maximum possible polymerase rate.

Finally, each time a new nucleotide is added by a polymerase, there is a risk that the nucleotide will be of the wrong sort. The primary driving force discriminating properly base-paired nucleotides from improperly paired nucleotides is the formation of hydrogen bonds between the incoming nucleotide and the corresponding nitrogenous base of the template nucleic acid. This process can be modeled by a similar Boltzmann distribution as used for determining the probability of a nucleotide becoming deactivated, but the ΔG^0 of base-pair hydrogen bond formation is approximately half that of spontaneous triphosphate hydrolysis. So, in the model the probability of an improper nucleotide addition event is calculated using $\frac{1}{2}$ the simulation temperature.

Recently, research has indicated that hydrogen bond formation is not sufficient to completely explain the fidelity of correct pair formation by nucleotide polymerases. It has been suggested that polymerases also take advantage of geometric differences between proper and improper pair formation to improve their ability to discriminate. In the model, this additional factor is treated as a simple flow problem. If a polymerase is roughly represented as a cylindrical tube, the narrower the tube is the greater will be its ability to reject geometrically unfavorable nucleotide pair configurations. However, the rate of the polymerase will also be affected by the radius of the tube. If rate is taken as the flux of nucleic acid through the tube, then we end up with a simple square root relationship between the ability of the polymerase to reject based on geometry and the number of nucleotides it can add in one simulation time-step. Specifically

$$\Gamma = \sqrt{r}$$

describes the geometric discrimination and

$$P_{error} = K * \Gamma$$

is the probability of making a mistake during each nucleotide addition event, calculated as the product of the temperature dependent Boltzmann distribution, K, and the geometric discrimination function, Γ.

Appendix A

Source Code

Following is the Ruby source code for the four main classes and the executable Ruby script used to run simulations based on YAML input files.

constants.rb

```
# Copyright (c) 2009 Joshua Ballanco
#
# constants.rb
#
# This file contains constants used elsewhere in the evolver project.

MAX_TOL_MUT_RATE = 0.34
MAX_POLY_RATE = 10
MIN_POLY_RATE = 1
```

environment.rb

```
# Copyright (c) 2009 Joshua Ballanco
#
# class Environment
#
# Abstract: The Environment class contains the entire simulation. It is
# primarily responsible for tracking all of the organisms in the simulation,
# calculating the density dependent death probability, randomly culling
# organisms according to that probability, and stepping each organism at each
# time step of the simulation

# The GenomeForEnvironment class is used to seed the starting population of
# the environment.
GenomeForSpecies = Struct.new(:genome, :population_frequency)

class Environment
  attr_reader :temperature

```

```
# The Environment must be initialized with the temperature of the simulation
# (in units of h-bond energy), the maximum and starting populations, and a
# GenomeForSpecies array enumerating the genomes to be use in creating the
# initial organisms.
def initialize(temperature,
               max_population,
               starting_population,
               *genomes_for_environment)

  # Some simple sanity checks on passed in arguments
  unless genomes_for_environment.inject(0) {|total, gfe|
    Rational(total + Rational(gfe.population_frequency))
  } == 1.0
    raise ArgumentError, "population frequencies of genomes must total 1"
  end
  if starting_population > max_population
    raise ArgumentError, "The starting population must be less than the
                          maximum population"
  end

  @temperature = temperature
  @max_population = max_population
  @organisms = []

  # Populate the environment
  genomes_for_environment.each do |genome_for_species|
    (starting_population * genome_for_species.population_frequency)
    .round.times do
      @organisms << Organism.new(genome_for_species.genome.dup, self)
    end
  end
end

# Runs the environment for iterations steps (default is 1000).
def run(iterations=1000)
  iterations.times { step }
end

# Set the number of threads to use, if we want to run the simulation
```

```
  # threaded.
  def use_threads(num_threads)
    @num_threads = num_threads
  end

  # Before stepping the environment, calculate the probability that any
  # individual organism will die due to resource constraints. This is modeled
  # as a agregate probability of $\frac{1}{(N-n)+1}$, evenly distributed over
  # the organisms in the environment, where $N$ is the carrying capacity of
  # the environment (max_population) and $n$ is the number of organisms
  # currently in the environment. At each step, determine whether or not to
  # cull an organism this round. If yes, then choose one to remove from the
  # environment at random.
  def step
    if (@num_threads && @num_threads > 1)
      @organisms.threadify(@num_threads) do |organism|
        organism.step
      end
    elsif RUBY_ENGINE =~ /macruby/
      @organisms.each do |organism|
        group = Dispatch::Group.new
        group.dispatch(Dispatch::Queue.concurrent) do
          organism.step
        end
      end
      group.wait
    else
      @organisms.each(&:step)
    end

    # This is the probability that 1 organism will die
    while (rand < (1.0 / ((@max_population - @organisms.length) + 1)))
      @organisms.delete_at(rand(@organisms.length))
    end
  end

  # The add_organism method attempts to add an organism to the environment.
  # If there adequate capacity, the organism is added and the method returns
  # true. If the environment is currently full, then nothing is done and
  # the method returns false.
  def add_organism(organism)
    if @organisms.length < @max_population
      @organisms << organism
      return true
    else
      return false
    end
  end

  # The report method returns a hash containing the values for this
  # environment as well as the results of iterating over the @organisms
  # array and calling each organism's report method in turn.
  def report
    { :temperature => @temperature,
      :max_population => @max_population,
      :current_population => @organisms.length,
      :organisms => @organisms.collect{|organism| organism.report}
    }
  end
end

# vim:sw=2 ts=2 tw=78:wrap
```

organism.rb

```
# Copyright (c) 2009 Joshua Ballanco
#
# class Organism
#
# Abstract: This is the base class for evolving organisms. It represents an
# abstract organism which is replicating its genome using a DNA polymerase
# and, when finished making the copy, replicating into two new organims.
#
# The Organism class is implemented as a finite state machine with the
# following states:
#   -- replicate_genome:  The organism is synthesizing a new genome using its
#                         existing genome as a template
#   -- divide:            Split into two, adding a new organism to the
#                         environment (if capacity for it exists)
```

```

class Organism
  # Initialization consists of setting the genome, translating a polymerase
  # from that genome, and passing in a reference to the environment in which
  # this organism lives.
  def initialize(genome, environment)
    @genome = genome
    @environment = environment
    @polymerase = @genome.translate_polymerase(@environment.temperature)

    # We’ll start with replicating the genome:
    @next_step = method(:replicate_genome).to_proc
  end

  # Step this organism and set the next step to the return value
  def step
    @next_step = @next_step.call
    return self
  end

  # We’re in the middle of creating a new genome. To do this, we allow the
  # polymerase to add as many nucleotides as it will. Following that, we query
  # the polymerase as to its current status, and proceed based on that
  # information.
  def replicate_genome
    @polymerase.add_nucleotides
    case @polymerase.status
    when :polymerizing
      return method(:replicate_genome).to_proc
    when :finished
      return method(:divide).to_proc
    end
  end

  # In order to divide, we first extract the new genome. If the new genome has
  # too many errors, then we reset and try again. Then, we create a new organism
  # from the genome and attempt to insert it into the environment. If there is
  # no room, we keep trying until there is (or we are randomly killed). Once
  # we’ve put the new organism in the environment, reset and start replicating
  # again.
  def divide
    @new_genome ||= @polymerase.new_finished_genome
    unless @new_genome
      @polymerase.reset
      return method(:replicate_genome).to_proc
    end

    @new_organism ||= Organism.new(@new_genome, @environment)
    if @environment.add_organism(@new_organism)
      @new_genome = nil
      @new_organism = nil
      @polymerase.reset
      return method(:replicate_genome).to_proc
    end

    return method(:divide).to_proc
  end

  # The report method returns details about the organism's genome and
  # polymerase
  def report
    { :genome => @genome.report,
      :polymerase => @polymerase.report }
  end
end

# vim:sw=2 ts=2 tw=78:wrap
```

genome.rb

```
# Copyright (c) 2009 Joshua Ballanco
#
# class Genome
#
# Abstract: The genome class comprises the genome of an organism. It is
# initialized with a length and some information about the sort of polymerase
# enzyme that it codes for. As it is copied (by a polymerase), it tracks
# progress and a count of the errors made. When requested, the genome can
# generate a polymerase or a copy of itself.
```

```

class Genome
  attr_reader :length, :added_nucleotides, :errors

  # Initialization requires the length of the genome as well as the rate and
  # directionality of the polymerase coded for by the genome.
  def initialize(length, polymerase_rate, directionality)
    @length = length
    @polymerase_rate = polymerase_rate
    @directionality = directionality
    @added_nucleotides = 0
    @errors = 0
  end

  # This method gets called during a Genome#dup. The length of the original is
  # left unchanged, but the properties of the polymerase generated are
  # recalculated based on how much mutation there was during replication.
  def initialize_copy(orig)
    high_dev = MAX_POLY_RATE - @polymerase_rate
    low_dev = @polymerase_rate - MIN_POLY_RATE
    max_dev = (MAX_POLY_RATE - MIN_POLY_RATE) / 2.0
    mut_frac = (@errors / @length.to_f) / MAX_TOL_MUT_RATE
    change_in_rate = (mut_frac * max_dev).round

    if change_in_rate > high_dev
      @polymerase_rate -= change_in_rate
    elsif change_in_rate > low_dev
      @polymerase_rate += change_in_rate
    elsif rand(2) == 0
      @polymerase_rate -= change_in_rate
    else
      @polymerase_rate += change_in_rate
    end

    if (@polymerase_rate > MAX_POLY_RATE || @polymerase_rate < MIN_POLY_RATE)
      raise RuntimeError, "Polymerase Rate out of Bounds"
    end

    # At the end, we reset @added_nucleotides and @errors to 0
    self.reset
  end

  # Start over replicating the genome (i.e. if an unviable copy was made and
  # discarded).
  def reset
    @added_nucleotides = 0
    @errors = 0
  end

  # Add a new nucleotide to the genome replica. If there was an erroneous
  # inclusion, add to the number of errors as well.
  def add_nucleotide(error = false)
    @errors += 1 if error
    @added_nucleotides += 1
  end

  # Seed a new polymerase with the directionality and rate defined by this
  # genome.
  def translate_polymerase(temperature)
    Polymerase.new(self, @directionality, @polymerase_rate, temperature)
  end

  # The organism can tolerate up to 1/3 of its nucleotides being mutated. It
  # would also not be viable if it wasn't finished being duplicated.
  def viable?
    if @added_nucleotides >= @length && @errors < (MAX_TOL_MUT_RATE * @length)
      true
    else
      false
    end
  end

  # The report method returns a hash of properties about the genome
  def report
    { :length => @length,
      :added_nucleotides => @added_nucleotides,
      :errors => @errors }
  end
```

```
end

# vim:sw=2 ts=2 tw=78:wrap
```

polymerase.rb

```
# Copyright (c) 2009 Joshua Ballanco
#
# class Polymerase
#
# Abstract: The polymerase class describes a generic nucleotide polymerase.
# When asked to, it will add a number of new nucleotides to the nascient
# genome. The directionality of insertion depends on the directionality that
# the polymerase was initialized with. Each polymerase will always be in one
# of two states:
#   -- polymerizing:  The genome is not yet finished.
#   -- finished:      The genome has been completely replicated.

class Polymerase
  attr_reader :status

  def initialize(genome, directionality, rate, temperature)
    @status = :polymerizing
    @genome = genome
    @directionality = directionality
    unless (@directionality == :forward || @directionality == :reverse)
      raise ArgumentError, "directionality must be :forward or :reverse"
    end
    @rate = rate
    @temperature = temperature
  end

  def add_nucleotides
    if @genome.added_nucleotides > @genome.length
      @status = :finished
      return
    end
    @rate.times do
      thermal_prob = Math::E**(-1.0 / @temperature)
      if ((@directionality == :forward) &&
         (rand < thermal_prob**2))
        next
      elsif ((@directionality == :reverse) &&
             (rand < (thermal_prob**2 * (MAX_POLY_RATE - @rate + 1))))
        return
      else
        @genome.add_nucleotide(rand < (thermal_prob * Math.sqrt(@rate)))
      end
    end
  end

  # If we're done polymerizing and the genome that we've produced is viable,
  # then return the genome. Otherwise, return nil
  def new_finished_genome
    if @status == :finished && @genome.viable?
      @genome.dup
    else
      nil
    end
  end

  # Resets the polymerase back to starting with a fresh genome to start
  # polymerizing anew.
  def reset
    @genome.reset
    @status = :polymerizing
  end

  # The report method returns a hash of properties about the polymerase
  def report
    { :status => @status,
      :directionality => @directionality,
      :rate => @rate }
  end
end

# vim:sw=2 ts=2 tw=78:wrap
```

evolver

```
#!/usr/bin/env ruby1.9
#
# Copyright (c) 2009 Joshua Ballanco
#
# evolver - This is the command line utility to manipulate the Evolver toy
# model.

$LOAD_PATH << File.join(File.dirname(__FILE__), ’..’, ’lib’)

require "optparse"
require "yaml"
require "evolver"

options = {}
OptionParser.new do |opts|
  opts.banner = "Usage: evolver [options] <environment file>.yml"

  options[:iterations] = 1000
  opts.on("-n [iterations]",
          "Number of iterations to run per environment") do |iter|
    options[:iterations] = iter.to_i
  end

  options[:report_frequency] = 10
  opts.on("-f [frequency]",
          "Report frequency (iterations between reports)") do |freq|
    options[:report_frequency] = freq.to_i
  end

  options[:snapshot_frequency] = 0
  opts.on("-s [frequency]",
          "Snapshot frequency (iterations between snapshots)") do |freq|
    options[:snapshot_frequency] = freq.to_i
  end

  options[:threads] = 1
  opts.on("-j [threads]",
          "Number of concurrent threads to run for calculations") do |thr|
    options[:threads] = thr.to_i
  end

  options[:report_values] = %W( forward_num
                                reverse_num
                                forward_rate
                                reverse_rate
                                organism_num
                                organism_rate )
  opts.on("-r [Value1,Value2,Value3]", Array,
          "Values to report") do |values|
    options[:report_values] = values
  end
end.parse!

def run(options, environment)
  genomes_for_env = environment[:genomes].collect do |genome|
    GenomeForSpecies.new(Genome.new(genome[:length],
                                    genome[:polymerase_rate],
                                    genome[:directionality]),
                         genome[:freq])
  end
  env = Environment.new(environment[:temperature],
                        environment[:max_population],
                        environment[:starting_population],
                        *genomes_for_env)

  env.use_threads(options[:threads]) if options[:threads] > 1
  # Run the simulation for the report frequency, then output the requested
  # values. Keep doing this until we’ve hit the max iterations limit.
  @iterations_complete = 0
  report(environment[:name], env, true, *options[:report_values])
  while (@iterations_complete + options[:report_frequency] < options[:iterations])
    env.run(options[:report_frequency])
    report(environment[:name], env, false, *options[:report_values])
    if (options[:snapshot_frequency] > 0 &&
        @iterations_complete % options[:snapshot_frequency] == 0)
      File.open(environment[:name] + ’_snapshots.txt’, ’a’) do |snapshotfile|
        snapshotfile << YAML::dump(env.report) << "...\n"
      end
    end
    @iterations_complete += options[:report_frequency]
  end
  env.run(options[:iterations] - @iterations_complete)
  report(environment[:name], env, false, *options[:report_values])
end

def report(name, env, print_header, *report_values)
  File.open(name + ’_out.csv’, ’a’) do |outfile|
    outfile << report_values.join(’,’) << "\n" if print_header
    env_report = env.report
    values = report_values.collect do |value|
      send("report_#{value}".to_sym, env_report)
    end
    outfile << values.join(’,’) << "\n"
  end
end

def report_forward_num(env_report)
  env_report[:organisms].inject(0) do |total, org|
    total + (org[:polymerase][:directionality] == :forward ? 1 : 0)
  end
end

def report_reverse_num(env_report)
  env_report[:organisms].inject(0) do |total, org|
    total + (org[:polymerase][:directionality] == :reverse ? 1 : 0)
  end
end

def report_forward_rate(env_report)
  env_report[:organisms].inject(0.0) do |total, org|
    total + (org[:polymerase][:directionality] == :forward ?
               org[:polymerase][:rate] : 0)
  end / report_forward_num(env_report)
end

def report_reverse_rate(env_report)
```

```ruby
  env_report[:organisms].inject(0.0) do |total, org|
    total + (org[:polymerase][:directionality] == :reverse ?
               org[:polymerase][:rate] : 0)
  end / report_reverse_num(env_report)
end

def report_organism_num(env_report)
  env_report[:current_population]
end

def report_organism_rate(env_report)
  env_report[:organisms].inject(0.0) do |total, org|
    total + org[:polymerase][:rate]
  end / env_report[:current_population]
end

environments = YAML::load_file(ARGV[0])
environments.each do |environment|
  run(options, environment)
end

# vim:sw=2 ts=2 tw=78:wrap ft=ruby
```

Vita

Joshua Ballanco

Place of birth	Hazel Crest, IL
Date of birth	October 5, 1980
Education	Stevens Institute of Technology, Hoboken, NJ Doctoral Candidate in Chemical Biology expected date of graduation, September 2009
	Stevens Institute of Technology, Hoboken, NJ Masters of Science in Chemistry May 2008
	Stevens Institute of Technology, Hoboken, NJ Bachelors of Science in Chemistry with Honors May 2002