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 toy 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 toy 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 alpha 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 alpha 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 toy 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 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 strategies are either evolutionarily dominant or evolutionarily stable. 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 toy model consists of a number of individual organisms competing with each other in an environment. A number of generalizations and assumptions have been made in order to render the problem being investigated tractible, 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 precisly those that may be observed in the laboratory, the trends observed should hold true when transfered to the bench.

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. 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, including justifications for each decision made in the design.

Environment

A key defining feature of the 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 order to mimic, at least empirically, the sort of density dependant inhibition to growth that is observed in all cells as the approach the carrying capacity of their environment, 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:

Chapter 3

Experimental Results of Polymerase Modeling

Chapter 4

Analysis and Discussion of Model Results

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.

environment.rb

```ruby
# 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 * mut_frac).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)
  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