Bioinformatics 445

Project 1 Report

11 February 2019

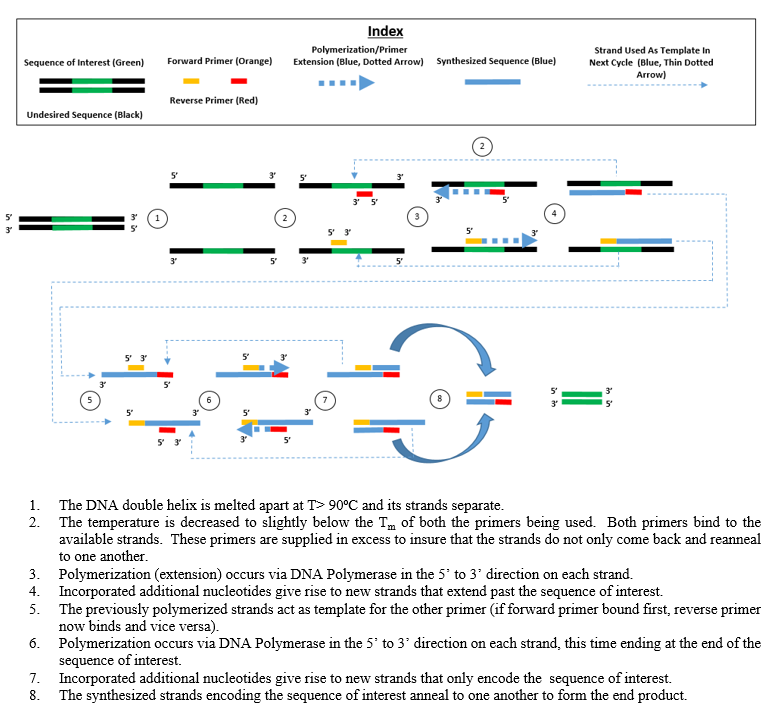
Group Members: Thomas Li

[TODO: come up with some believable-sounding stuff regarding role delegation]

[TODO: write and submit individual versions of this report]

Background:

Polymerase Chain Reaction (PCR) is a method for synthetic DNA replication in which primers, small segments of complementary nucleotide bases, are attached to separated DNA strands and extended to contain a desired subsequence, doubling its presence. This is repeated over several cycles with the resulting set of DNAs.



Forward strands are copied by binding a primer to the end of the sequence at extended towards the front, while reverse strands are copied by binding a primer to the start and extending it towards the back. The importance of this will be discussed later.

Implementation:

We represented the DNA as a class containing data members for the forward strand and reverse strand as well as methods to denature the pair (clear the reverse strand and return a new object containing it), bind a primer (given one empty strand and that the primer complement can be found), and extend the primer (given the presence of a primer strand shorter than the template strand). The object-oriented approach allows us to treat data objects as fully-defined, self-contained, abstract entities that are easier to define and manipulate then raw variables.

Regarding the internal logic of the methods, primers could only be bound if the entire complementary strand could be found in the template, indicated by the str.find() function returning a value greater than -1. This was mainly done for convenience as it took only one line of code to implement. The extension function adds the extension length onto the primer length instead of including it, not for any reason other than that I had initially written it that way and it worked. The denaturing function was based on the stack pop method, which provided a nice model for how to internally split data objects while keeping them cleanly separated and organized.

The experiment at large involves parameters for template strand length, primer length, extension length, and primer count. A list contains all DNA objects in the experiment, starting with a pair containing the randomly generated template strand and its complement. Denatured strands are appended to the list, which is cycled through to perform bindings and extensions on all strands until a stop condition is reached. A function collects data from the experimental output, including the number of strands, length distribution, and amplification of the target sequence.

Due to the single purpose of the program, this is currently all defined globally, though this may change should we reuse the code. The global scope also contains two general helper functions: one to randomly generate a strand of DNA from bases and the other to get the complement of a strand of DNA. The latter is called several times throughout the program, including in the binding and extension functions.

The final results of the experiment are written to a text file so that they can be saved, exported, and reformatted more easily.

Difficulties:

Even though I've never programmed in Python before, I already knew most of the basic programming concepts through working with C++ and JavaScript and substituting the syntax was relatively easy for the most part, though there were a minor hiccups from getting used to immutable strings, having to capitalize "true" and "false", and keeping track of types.

In terms of language-independent difficulties, I had accidentally allowed empty strands to be extended even if no primer had been bound. I noticed a problem after more strands were being generated than the amount of primers a large number of strands were being generated at minimum length. After two days, I found that I'd written "and" instead of "or" for the condition checking if primers were present or not.

Results:

Given a template of 2000 bases, a target sequence of 200 bases, primers of 20 bases each, polymerase processivity of 200+/-50 bases, the following length distribution resulted from extending 100000 primers:

[TODO: run the program again and make a histogram from the output file]

The final output contained 100,002 strands from 20 complete cycles and one additional partial cycle, of which 70,053 or around 70% contained the target sequence. The average strand contained 195 bases.

Given that the target subsequence is 200 bases long, all strands containing the sequence would have to be 200 bases long or longer. Given that the number of strands 200 bases or longer is equal to the number of strands containing the target (70,053), we can also infer that all strands 200 bases or longer contain the target, and that all strands exactly 200 bases long are exact copies of the target sequence with no excess bases.

Interpretation:

The final output contains 70,026 strands of exactly 200 bases, which means that, of the 100,000 primers used, just over 70% resulted in the creation of the exact desired subsequence, illustrating the relative efficiency of the process.

The overwhelming large-scale clustering of strands at length 200 comes from the usage of alternating forward and reverse primers, which results in the following properties:

- An under-extended forward strand cannot be propagated because it lacks one or bases at the end of the sequence required for the reverse primer to be bonded, and vice versa.

- An over-extended forward strand cannot result in over-extended reverse copies because the reverse primer binds to the end of the sequence and extends towards the front, ignoring the excess bases and being capped by the proper start of the sequence, and vice versa.

Given these, only the replication of strands that are exactly the length of the target subsequence is encouraged, the effect of which is amplified after several cycles.

[TODO: add some stuff about gel]