PCR Primers and Machine Learning

Michael Kuehl and Andrew Wittrock

August 8, 2001

Contents

1 Description of Problem

The problem set before us is one of machine learning. How do you get a computer to recognize patterns in data that seem nonintuitive and only reveal themselves after exhaustive study by a human mind? The biology of the project is complex, but it is not necessary to understand the intricacies of DNA replication and primer design to create a system to classify primers. What we want is a software package that will train on a set of primers (represented by character strings) and their associated types, and then be able to predict the types of primers not in the training set.

Our training set was provided by the Schnable lab at Iowa State University. We assume that the primers in the training set were designed with the best available technology. We are building a bolt-on post processor to select among primers designed to amplify a given target or to prioritize among multiple targets.

1.1 Experiment Setup and Baseline Algorithm

As an automaton for the evolutionary algorithm, a finite state machine (FSM) with 64 states was chosen. These FSMs consisted of an initial output/state pair and a set of n states. Each state consisted of a set of output/transition pairs, one for each possible input value. For example, in the FSM in Figure 1, the FSM starts in state 23 with an initial output T. If the FSM is in state 3 and a G is input, the FSM will output an A and go to state 7.

```
--------------
States:64.
Start:T->23
   If C |If G |If A |If T
---------------------------
 0)A->24|A->14|G->17|C->21
 1)A-> 1|A->28|C-> 6|A->28
 2)A-> 1|A-> 3|A-> 2|C->26
 3)G->18|A->18|A-> 7|A->22
 4)A-> 6|A-> 8|C->17|A->24
 ...
--------------
```

Figure 1: A finite state machine

To run an FSM on a given primer, the FSM is first reset and then the primer is input into the FSM one base at a time. The outputs C, G, A, and T are translated into 0, 1, 2, and 3, respectively and these numbers are used to classify the primers: 0 = good primer; 1 = bad primer, one band; 2 = bad primer, two bands; 3 = no guess. The fitness of a given FSM is calculated using the reward matrix given in figure 2, where the rows represent what the primer actually is, and the columns represent the output of the FSM.

$$\begin{pmatrix} 3 & -4 & -4 & 0 \\ -5 & 15 & 3 & 0 \\ -5 & 3 & 7 & 0 \end{pmatrix}$$

Figure 2: Reward matrix

For example, if the primer is good, but at a certain base, the FSM guesses that it is bad with one band, then it would receive a reward of -4 for that guess. The FSM's fitness is calculated by summing all of the rewards over all of the bases of all of the primers in the training data.

The method of evolution chosen was single tournament selection with tournament size four. Each generation, the population of FSMs is shuffled randomly and divided into groups of four. Each group of four is then sorted by fitness. The two most fit FSMs are then copied over the two least fit FSMs, and these two copies are crossed with each other and mutated, and then their fitness is calculated.

The crossover operator used was two point crossover. Two states are chosen at random and all of the states between them are swapped with the corresponding states in the other FSM.

Single point mutation was used. 10% of the time, the initial state or initial output action was mutated. 30% of the time, a state transition was mutated. 60% of the time, an output action was mutated.

The evolutionary algorithm was run with a population size of 600 for 1000 generations. 100 runs like this were performed, and the best creature from the last generation of each run was saved.

2 Experiments and Results

2.1 Hybridized Fitness

The natural fitness function would be the number of 'right' guesses, divided by the number of 'wrong' guesses. This performed very poorly. The problem is that machines that always output 'type 1' as their guess are very easy to find, and get pretty good fitness. This was the motivation for the incremental fitness function used in the original code. After some thought, it was theorized that it would be beneficial to switch back to the 'right vs. wrong' fitness function after the incremental fitness function had dragged the population out of the quagmire of 'always guess type 1.'

Another version of the hybridized fitness function was tried, using a 3 by 3 matrix to weight the FSM output in fitness calculation. The matrix chosen was simply the absolute value of the payoff matrix for the incremental fitness function.

The hybridized fitness function performed better than the baseline code. There was no substantial difference in performance between weighted and unweighted hybrid fitness functions.

2.2 Indecision Penalty

In the original algorithm, there is no fitness penalty for not reaching a conclusion (in other words, the FSM has the option of outputting "I don't know" as opposed to guessing the type of the primer). Indecision penalties of -1 and -2 were tried.

The performance of the baseline code was not affected by the inclusion of a penalty for indecision.

2.3 Half States

In this experiment, the number of states in the FSMs were halved from 64 to 32. The motivation for this change was the concern that 64 states were too many for this problem, and that evolution was overtraining the FSMs, causing them to memorize the data set rather than finding patterns. The baseline algorithm was used with the number of states being the only difference.

The performance of the 32 state FSMs was inferior to that of their 64 state counterparts. It does not seem to be the case that the FSMs are overtraining on 64 states.

2.4 Cross-validation

In this experiment, ten percent of the training data was set aside and the baseline algorithm was run on the remaining ninety percent. The goal for this experiment was to verify that the FSMs were actually finding patterns rather than simply memorizing the training data. After the algorithm finished, the resulting best of run machines were tested on the ten percent of the data they had not been trained on.

The results of this experiment verified that the FSMs were actually finding patterns in the training data. Even though they had never seen the ten percent of the training data that was set aside, the FSMs classified them with the same amount of accuracy as the ninety percent that they were trained on.

2.5 Ramped Fitness

In this experiment, we altered the fitness function by scaling the payoff matrix depending on what position in the primer the FSM was currently reading. The goal for this experiment was to see if we could smooth out the fitness landscape even more. The payoff values were multiplied by

$$\frac{(x/3)^2}{1+(x/3)^2} \tag{1}$$

where x is the position in the genome.

The results for this experiment were not much different from those of the baseline algorithm. The FSMs in this experiment were very nearly just as successful in identifying good and bad primers as the FSMs in the baseline algorithm.

2.6 Hybridization

In this experiment, we seeded an initial population of 600 FSMs with 100 evolved FSMs from the ramped fitness function experiment. The ramped fitness function was used for this experiment as well. Evolution was again run for 100 runs of 1000 generations and the 100 new best of run FSMs were fed into another initial population for a second level of hybridization. The goal of this experiment was to see if blending evolved FSMs from different populations would improve the quality of the FSMs.

The results of this experiment showed that hybridization did in fact improve the quality of the FSMs and also very quickly found an optima. After one level of hybridization, the average fitness of the new best of run FSMs was about the same as the maximum fitness of the original best of run FSMs. However, the second level of hybridization failed to yield similar improvements. Every run improved only slightly and the fitness of the best FSM in all runs converged to a single value, implying that evolution had found an optima.