David Chen

I’m a 6th year graduate student in Bioengineering interested in the design of biological circuits. I’m interested in learning more about parallel programming, especially in topics that are useful to companies today. For the sample application, I will be discussing a problem that some members in my lab have been working on for the past couple years.

Thanks to decades of molecular biology research, we are starting to understand the rules that control gene expression. One of the critical properties one would like to predict is the expression level: ie, given a DNA sequence, can we predict the amount of protein being produced. Figure 1 illustrates the process of gene expression.

Figure 1. The central dogma of biology is that DNA is transcribed to mRNA, and mRNA is translated to protein molecules. Generally, we are interested in the amount of protein molecules being produced. In this project, we focus on the ability to predict translation strength (also referred to as the translation strength).

We have recently developed in the Arkin lab a methodology to test the translation efficiency of an arbitrary sequence of DNA. The goal is to design a large number of DNA sequences that we synthesize, and then we set up the experiment so that transcription is the same between each DNA sequence. We then measure the protein concentration for each DNA sequence, so we can measure the translation efficiency.

The approach in our lab to exploit this high-throughput experimental pipeline is to design sequences with desired characteristics, and then perform a multiple linear regression of expression strength versus features.

Table 1. Example of features and the # of levels for each feature that we wish to explore. Generally, we wish to force each feature into three bins of low, medium, or high. In this example with three features with three levels each, we wish to design $3^3=27$ DNA sequences (before any replicates).

<table>
<thead>
<tr>
<th>Name of feature</th>
<th># of levels</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codon adaptation index</td>
<td>3</td>
<td>How well the distribution of codon usage matches the host organism. A codon refers to a triplet of DNA that corresponds to an amino acid (proteins are composed of amino acids).</td>
</tr>
<tr>
<td>Min free energy</td>
<td>3</td>
<td>How well the mRNA can fold into a structure. Computed via existing RNA-folding packages.</td>
</tr>
<tr>
<td>%GC content</td>
<td>3</td>
<td>The fraction of nucleotide bases that are G</td>
</tr>
</tbody>
</table>
Table 1 gives an example of the types of features we are designing for. We wish to design sequences that exhaustively search all possible combinations of all levels for the features. Figure 2 illustrates an overview of the program’s control flow. Initially, we start the solution database with one seed DNA sequence. We then randomly choose a desired design goal that specifies the level for each of the features. We look in the solution database for a sequence that is close to our desired combination, and then begin the sequence evolver. In each iteration, we mutate the sequence and ask if the new sequence matches the desired design goal. If true, we store the current sequence; otherwise, we continue evolving for up to 1000 iterations. Notably, the sequence evolver routine is easily parallelizable since it does not need to communicate globally until it has either found a solution it needs to store or it has given up.

Figure 2. Overview of the sequence design process. A target combination of feature-levels is selected, and then a sequence from the solution database that is close to the desired features is chosen. A sequence evolver process is run for up to 1000 iterations. If the sequence is mutated to match the desired levels of features (the current design goal), this sequence is stored into the solution database.

Finally, we note that we wish to start with N seeds, where each seed represents a starting DNA sequence that is “far” from the other seed sequences. The application is currently run on the JBEI computing cluster (82 dual-core AMD Opteron processors, 600GB aggregate memory). Each of the N seeds is also embarrassingly parallel from the other seeds since there is no reason for communication between processes that have started from different seed sequences. In the current implementation, the users evenly divide the N seed sequences among the available cores. Ideally, we would like all N seed sequences to finish designing at the same time, but in practice, we find that some seed sequences are extremely difficult to
achieve the desired combinations of feature-levels and take much longer to run. One possible improvement in the process is to dynamically re-allocate the computing power given to each seed sequence based on the difficulty the solver experiences.

Overall, the procedure achieved the objective of designing DNA sequences with the desired combinations of feature-levels. The optimization framework and evolver routines are written in Python using a distributed memory platform. Some of the computations require calling 3rd party tools. The JBEI cluster is not on the top 500 list. A key challenge with the current design is that the clock run-time is tremendous. In the latest experiment with 50 seeds and ~30 feature-level combinations, execution took almost a month on the JBEI cluster. The application scales well to more processors if we also want more seed sequences to design for (ie, weak scaling). Because of the previously discussed issue regarding some seed sequences that are extremely difficult to design for, the system does not scale strongly. The user ends up waiting at the end for these difficult seed sequences, and there is no assurance that we cut down the waiting time by increasing the number of processors in the current scheme.

Detailed information about the process is available in a recent publication, http://bioinformatics.oxfordjournals.org/cgi/pmidlookup?view=long&pmid=24398007