UCSC BME 205 Fall 2013

Intro to Bioinformatics
Degenerate codons

(Last Update: 20:52 PST 26 November 2013)

Cassette Mutagenesis and Degenerate Codons
Due 6 Dec 2013

This assignment is intended to deepen your understanding of cassette mutagenesis, the genetic code, and libraries of mutant genes.

http://www.biology-online.org/dictionary/Cassette_mutagenesis defines "cassette mutagenesis" as "*The production of mutants within a region (often bounded by unique restriction sites) by the use of synthetic oligonucleotides that fill the gap with mutants designed into the synthetic genetic material.*" It is a popular method for doing site-directed mutagenesis.

In a typical application, a segment of a gene is excised with restriction enzymes and a synthetic DNA sequence of the same length is inserted in its place. Because synthesis and purification of small (15–50bp) DNA strands is very cheap (around $0.20–$20/base-pair, depending mainly on quantity needed, but also on oligo length, purification technique, and corporate markup), cassette mutagenesis is a very low-cost way to modify genes, though multiple modifications are limited to being close to each other in the sequence.

There is a nice 33-page review article on gene synthesis techniques:
Randall A. Hughes, Aleksandr E. Miklos, Andrew D. Ellington,
Chapter twelve—Gene Synthesis: Methods and Applications
Methods in Enzymology, 498:277-309, 2011.
You don't need the information in it to do this homework, but it is worth reading if you ever need to think about synthesizing genes.

One of the interesting aspects of DNA synthesis is that you are not limited to creating just one synthetic sequence. You can synthesize *DNA libraries*: pools of DNA sequences that have a controlled amount of variation. In DNA synthesis, as each base of the chain is added, you have a choice of 4 reagents to add the 4 different bases. You can also use a mixture of reagents, so that some of the chains being synthesized get one base and some get another. If you are doing in-house synthesis, you can adjust the ratios of the reagents to control what fraction of the sequences get each base. Most commercial synthesis companies limit you to specifying which bases are allowed at each position, but then synthesize roughly equal numbers of chains with each base.

There are other techniques for getting codon-specific variation in randomized DNA. See for example, Entelechon's page about their codon-precision mutant libraries, which use codon-specifc synthesis techniques to get a finer level of control on the random lbrary. Here, we're just looking at what can be done with degenerate codons.

To specify the set of bases desired, people use an enhanced alphabet, with 15 different letters, representing the 15 non-empty sets of the 4 bases:

code	stands for	code	stands for

A	adenosine	M	A C (amino)
C	cytidine	S	G C (strong)
G	guanine	W	A T (weak)
T	thymidine	B	G T C
U	uridine	D	G A T
R	G A (purine)	H	A C T
Y	T C (pyrimidine)	V	G C A
K	G T (keto)	N	A G C T (any)

from http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml

Thus the DNA sequence ARDCN represents a set of 24 different sequences:
A-[A or G]-[A or G or T]-C-[A or G or C or T] Note that U occurs only in RNA and T only in DNA, so there are only 15 different degenerate bases, not 16, as U and T are taken to be synonymous (and the empty set is not a possible degenerate base). For this assignment, use the DNA letters, not the RNA ones.

Since the cassette mutagenesis is most often done in protein-coding regions of genes, it is convenient to think of the mutations in terms of codons in the frame used for translation. With 15 different sets possible at each of the three positions of the codon, there are 15^3=3375 different degenerate codons that can be expressed, rather than the 4^3=64 standard codons. Each degenerate codon now represents a set of possible DNA sequences, and gets translated to a set of amino acids.

The primary goal of this assignment is to produce a table containing all possible sets of amino acids (including STOP) that can be encoded by a single degenerate codon, and the "best" degenerate codon to use for that set.

First, it should be obvious that not all sets of amino acids can be represented by degenerate codons, as there are 2^{20}=1048576 (or 2^{21}=2097152 if we include STOP) such sets, and only 3375 degenerate codons. Furthermore, several degenerate codons will translate to the same set of amino acids (for example, CTT, CTN, YTR all translate to just leucine). How many distinct sets of amino acids *can* be represented by a single degenerate codon? (STOP is a legal possibility here, so "*W" produced by TGR is different from "W" produced by TGG.)

Probably the first thing to do is to create a dictionary that maps each degenerate codon to the set (or, better, to a multiset that includes multiplicity) of amino acids it codes for. I found it convenient to first make a dictionary mapping each codon or degenerate codon to the codons that it can expand to. Then I translated each codon to an amino acid and used a dictionary to represent the multiset. The mapping of degenerate bases to the bases they represent is fixed, and can be hardwired in the code, but the genetic code does vary, so there should be an option to read it from a file.

Bonus point: give people the option of specifying one of the genetic codes from ftp://ftp.ncbi.nih.gov/entrez/misc/data/gc.prt and read that file directly from the web (optionally, from a local copy). Your parser for the file need not be very robust, but should at least know to ignore comments. You may want to use the urllib module for fetching the file, as it is simpler to use than the ftplib module for just reading data.

Next, you'll probably want to turn things around to make a dictionary mapping sets of amino acids to lists of degenerate codons that produce that set. Since "set" and "dict" are not hashable types in Python, you can represent sets of amino acids as strings with the amino acids in alphabetical order by one-letter codes, using

* for stop. If you have a multiset represented as a dict, say amino_c["W"]=1 amino_c["G"]=1 amino_c["R"]=2, which would be the multiset for NGG, then `''.join(sorted(amino_c.keys()))` would give you an appropriate string that can be used as a key. (Note: "frozenset" is hashable, if you find that easier to work with than strings.)

At this point, you can easily determine how many different unique sets of amino acids there are, and what (degenerate) codons code for each set. How many distinct sets are there that don't include a stop codon?

Because most sets will be represented by multiple degenerate codons, it would be nice to have a table that for each set gives the "best" degenerate codon to use. Deciding which codon is best is a non-trivial task, as a careful analysis would look at what codons translate best in the host system that will do the translation and also try to avoid creating stable secondary structure in the messenger RNA.

For this assignment, we'll look at simpler versions of this problem, defining the goodness of a degenerate codon in two ways. First, we'll not worry about expression levels at all, but just about how well a library of DNA sequences built from the degenerate codons samples the amino-acid sequences. In the example above, the codon NGG produces "R" half the time, "G" a quarter, and "W" a quarter. We could get a more uniform sampling if we used BGG, which produced each of the three amino acids a third of the time.

Let's define "imbalance" as the maximum difference between the number of codons for each amino acid represented by the degenerate codon, divided by the total number of codons represented. That is, something like this

```
def imbalance(dcodon):
    counts = amino_counts[dcodon].values()
    return (max(counts) - min(counts))/float(sum(counts))
```

For example, "TTN" represents "F" with 2 different codons, and "L" with 2, for an imbalance of (2-2)/4=0. "YTN" also represents "FL", but "L" now has 6 codons and "F" has 2, for an imbalance of 0.5. "RTN" and "RTR" both represent "MIV", but "RTN" has 3 for I, 4 for V, and 1 for M, for an imbalance of 0.375, but "RTR" has 1 for I, 2 for V, and 1 for M, for an imbalance of only 0.25.

(1) Print a table that has one line for each amino-acid set, with 3 tab-separated fields: the amino-acid set (just the letters in alphabetical order, no punctuation), the minimal imbalance, and a (degenerate) codon with minimal imbalance. The table should have the lines in alphabetical order by amino-acid set. This is the "minimal" format.

Bonus point: replace the third field by *all* the degenerate codons that have minimal imbalance for that set (comma-separated). This is "min-codons" format.

We can still do more optimization of the choice of codons, when there are multiple degenerate codons with the same imbalance. Normally, when we are synthesizing genes, we want each synthesized gene to have a reasonably high expression. Although optimizing a gene for maximal expression is not trivial, we can do a fairly good approximation by looking at the codon frequency in the host organism for the gene (most often, *E. coli*). Those codons that occur frequently are usually translated faster than less frequent codons, resulting in more protein expression per mRNA molecule.

For example, if we want a codon for "L" for expression in *E. coli*, we might choose CTG, since it is the most commonly used codon for "L" in *E. coli* CFT073. [http://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=199310]. For degenerate codons, we can use the average frequency of all codons it expands to as a crude measure of codon preference. For example, if we want to get "FL", we have many choices with an imbalance of 0, but TTK has the largest average frequency (0.0186) and so we

would choose it. Note, TTW also has an average frequency of 0.0186 but is slightly less frequent than TTK, while YTC has only 0.0139, though it still has an imbalance of 0.

(2) Modify the program so that the codon selected is the highest frequency of the minimal-imbalance codons. This is still the "minimal" format. If you did the optional "min-codons" format, sort the codons so that the most frequent is first.

Bonus point: Modify the table above to print the amino-acid set then a list of all codons with their imbalance and frequency, in decreasing order of desirability (lower imbalance being more important than higher frequency). The lines should look like this:

```
ACGPRSW ['BSS, 0.08, 0.0159', 'BSK, 0.08, 0.0147', 'BSB, 0.11, 0.0151']
ACGRSTW ['DSS, 0.17, 0.0159', 'DSK, 0.17, 0.0130', 'DSB, 0.22, 0.0147']
```

This is the "full" format.

Bonus point: provide several different output formats and options to choose which one to use.

Bonus point: read the codon frequency table in a standard format (like that provided by http://www.kazusa.or.jp/codon/ for GCG programs), so that the host organism is easily changed. Even better, fetch the codon table directly from the web, without needing a local file.

Super bonus point: write a program that accepts a protein sequence and provides an optimized DNA sequence to be a gene for it. The protein sequence can have multiple amino acids in any position, like in regular expressions. For example, A[AEV][DN] represents six different amino-acid sequences: AAD, AAN, AED, AEN, AVD, AVN. For some combinations of amino acids, there will not be a degenerate codon that matches exactly. In that case, take one that adds minimally to the set of sequences generated. Note: I have implemented everything in this assignment *except* this super bonus point. There are undoubtedly subtleties that I've not thought about yet. Doing optimization for avoiding secondary structure in the mRNA is definitely beyond the scope of this class.

To turn in

Turn in the source code for the program and any modules you created for it. Also turn in one output using the standard genetic code and the codon frequencies from *E. coli* (http://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=199310).

Note: you must provide good user-level documentation for your program. I must be able to run your program easily and not have to guess what options to set. You get no bonus points for options that I can't figure out! Ideally, I should be able to run the commands

```
degenerate --output_format=full
degenerate --output_format=minimal
degenerate --output_format=all-codons
degenerate --output_format=min-codons
```

(if you implement all output formats, which is not required) to get output to stdout, without having to locate auxiliary files or change hard-wired file names. Note that there are **no inputs** for these commands—they just generate output to stdout.

Other arguments that you could recognize (with reasonable default values) include

```
--codon 'http://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=199310'
--code_table=ftp://ftp.ncbi.nih.gov/entrez/misc/data/gc.prt
--gene_code=Standard
```

Note that you can open a URL for reading as easily as opening a file with the urllib package.

I'd like the imbalance reported with {:.3f} format, so that I can compare the output files more easily. Look at the instructions for the sort order also.

Things learned for next year

SoE home

Kevin Karplus's home page

Biomolecular Engineering Department

BME 205 home page

UCSC Bioinformatics research

Questions about page content should be directed to Kevin Karplus
Biomolecular Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
USA
karplus@soe.ucsc.edu
1-831-459-4250
318 Physical Sciences Building