BIOL591: Introduction to Bioinformatics
Constructing and using a PSSM to find motifs in a genome

I. What do we need to construct a useful PSSM?

We now have a strategy to find NtcA sites in the genome of *Anabaena*: make a position-specific scoring matrix (PSSM) based on the aligned sequence of the region encompassing known NtcA-binding sites (in 71NpNsSm.txt[1]) and then use it to find regions within the genome that best match the PSSM. Let's take a look at the program, *FindMotif.pl*, that will be used to accomplish the task.

Download and open *FindMotif*. The header tells us that it's supposed to do what we thought it was going to do. It will use two input files, one containing the aligned sequences from which the PSSM will be constructed and the second the *Anabaena* genome. The header also tells us that the program is currently short circuited for testing purposes. Let's note that and run the program (of course you may have to adjust the file names to get the program to work).

If all went well, after several seconds of thought, the program should have produced a few screenfulls of sequences. The header claims that it also produced an output file. Find that file and take a look at it by reading the file into your favorite word processor. (You will have to adjust the margins and change the orientation to landscape to fit each line of output into single lines of display). If all went well, there should be sequences resembling bona fide NtcA-binding sites. (If you've forgotten what they look like, bring into another document 71NpNsSm.txt). (In both cases, it will help to change the font to something nonproportional, like Courier New, so that the T's and C's have the same width).

Evidently, all is ***not*** well, because while the set of proven NtcA-binding sites (71NpNsSm.txt) gives a pleasant alignment, as shown in Fig. 1A, the sequences produced by *FindMotif* look chaotic, as shown in Fig. 1B. Is it because this is only a testing version of the program (reading in only the first 50,000 nucleotides of the genome). Perhaps we'd see nicer looking sequences if the program considered the entire genome. It will cost you something less than 45 min to find out, once you eliminated the statement that confines the program to 50,000 nucleotides.

A
```
aatttagtatcaaaaataacaattca
cgttctgtaacaaagactacaaaact
aattttgtagctacttatactatttt
aaagctgtaacaaaatctaccaaatt
tcatttgtacagtctgttacctttac
aaatctgtaacatgagatacacaata
ATATCTGTAACATAAGCTACAAAATC
AAAATCGTAACAATTTATACGATTTT
ATTACTGTAACTAACTATACTAAATT
CGTTTGGTATCCAAGATAACTTTTTA
CCATCTGTAGGCTCTGTTACGTTTTC
```

B
```
CATTGGGTATGGACGGTAGCTTTTAG
ATATCAGTAACGATGTATTTGATTCA
ACAGGTGAGGCTAAGAAAAGCTAACA
TTATCTACAGGTAAATTCACAATTCC
CAAATTTAACGATGTTATTCGCATTT
CGATCGCTCCGCAATGCTCCATCTCT
ACATCAGCATCGATTGTCCCACTTCA
CATACGGTTCTGGTAATGACCCATTA
ATTGCCGTCACATTTTTGTAAAAACT
CTTTTGGTAAGGACACCAGACGGATC
GCAGCAGTGAAGACCGCACCATCCGC
```

Excerpted aligned sequences from (A) 71NpNtSm.txt and (B) output from *FindMotif*. Position of accepted consensus sequence is high-lighted in yellow.

SQ1. Remove the nucleotide restriction on the input file and rerun *FindMotif*. Are the output sequences more NtcA-like?

While that's going on, we can consider the alternative, that the program isn't as effective as we would like. With that in mind, consider the logic of the program by looking at the **Main Program**...

[1] Go to the Scenario main page to find links to *FindMotif.pl* and 71NpNtSm.txt.

... “Read motifs”... seems plausible... “Calculate PSSM”... you did something like that in class... “... considering only informational positions...” What does that mean? The pertinent problem was raised in the last set of notes (**Section III.D**). That section talks about a balance between specificity and comprehensiveness. How does the program strike this balance?

SQ2. Which section of *FindMotif* is responsible for reducing the number of positions within a window that is considered by the program? How does it do it?

You may have found a problem. How ***should*** the program go about reducing the number of positions, and how can you help it out? What we’d ***like*** to do is to write a loop that will calculate **H** over all nucleotides of a position, then calculate the information content for that position, and finally compare that value to some threshold value you determine. Those with information contents greater than the threshold will be considered in constructing the PSSM (How to pick a good threshold value? Later.)

SQ3. In general terms, what do you need in order to calculate the information content of each position?

SQ4. In specific terms, what variables in *FindMotif* have what you need?

II. Hashes

The preceding question should brought you face-to-face with the list of variables used by the program, and some of them may have looked strange. You probably recognized that `$contig` is a scalar variable that contains the sequence of a specific contig (i.e. a specific large sequence). You probably also recognized that `@contig` is an array variable that contains the sequences of multiple contigs. You knew this from the markers “$” and “@”. But what is `%count`?

Just as all scalar variables in Perl are preceded by “$” and all array variables by “@”, the marker “%” precedes all variables of a different class: ***hashes***. Hashes are very much like arrays in that a hash variable refers to a collection of values. They differ from arrays in that their collection is not ordered. You get to elements of an array by giving a numeric subscript: `$contig[0]` refers to the first element of the array, `$contig[1]` to the second, etc. Suppose, however, you want to refer to the number of sequences that contains A’s at a given position, the number of sequences that contain C’s at that position, etc. You could do this with four arrays:

```
my (@count_of_As, @count_of_Cs, @count_of_Gs, @count_of_Ts);
$count_of_As[$position] = ...
$count_of_Cs[$position] = ...
```
etc.

This approach will almost certainly lead to repetitive code, a line to calculate the number of A’s, a separate line to calculate the number of C’s, etc. It would be nice to be able to say:

```
my %count;                                  # Declares a single hash
$count{$nucleotide}[$position] = ...
```

meaning, the count of a given nucleotide at a given position is.... The problem with this approach is that while positions are numeric (ranging from 0 to the length of the motif – 1), nucleotides are not. They are (“A”, “C”, “G”, “T”), and to give a letter as a subscript of an array will not work.

Hashes allow letters or indeed any string to serve as the identifier for an element of an array. They are therefore often called unordered lists or associative arrays (since the value of the array is associated with the string – or key – that refers to the value).

Hashes can be populated (given values) element by element in the following way:

```
my %hash;
$hash{$key} = constant or scalar variable;
```

Note that hash references are encompassed in ***curly brackets*** not the square brackets required by arrays. Another way to initialize a hash is all at once:

my %hash = ("ala" => "alanine", "arg" => "arginine", "cys" => "cysteine", ...);

A list is assigned to a hash, consisting of paired keys and values, connected with the => arrow.

Hashes are great for problems that require manipulation of text (which certainly includes many bioinformatics problems). We've seen hashes before, in the program *BlastN.pl*, where a hash `%seen` is used to keep track of what matches have already been encountered by the program.

SQ5. Examine *BlastN.pl* and see how %seen and $seen{ } is used. Do you see how it can prevent the program from listing the same match twice?

III. Using a hash to calculate information content

To return to our problem with *FindMotif*, we need to find a way of calculating the information content at each position, requiring that we access the counts at each position for each nucleotide. Evidently, this information is kept in the hash `%counts`. The overall task at hand might go something like this:

Loop over all positions

  Loop over all nucleotides
    Calculate a fraction = (counts of the given nucleotide at the given position) /
                           (total number of motifs)
    Calculate fraction * log(fraction)
    Add that product to the sum that will eventually be $\mathbf{H_c}$
  End nucleotide loop

  Calculate information content from $\mathbf{H_c}$
  If the information content exceeds a given threshold, note that position

End position loop

SQ6. Try to rewrite the subroutine Determine_informational_positions so that it works.

We return to the problem of deciding on what would be a good value for the threshold. There's no help from theory. I advise that you print out the information content of each position and from the calculated values, choose a threshold that looks good to you. You might also try different values to see what sequences the program returns with each one.