Finding CRISPR arrays

Specification

Program Name: randomizedMotifSearch.py

Input/Output: STDIN, STDOUT

Options:

- -i iterations (int)
- -k motif length (int)
- -p pseudocount (float)

In [ ]:

```
Ex: python randomizedMotifSearch.py -i 1000 -k 13 -p 1 < input.fa > output.out

Inspection Notebook example:
main ( "input.fa", options = [ "-i 1000", "-k 13", "-p 1"] )
```

Background

CRISPR arrays exist in most archaeal species and many bacterial species. The arrays are encoded in chromosomal DNA as direct repeats (DR) that flank sequences (spacers) that provide immunity against invading plasmids, viral sequences, and other foreign DNA .. and RNA. These arrays can be short, comprising 1-2 spacers, or they can contain hundreds of spacer elements. The arrays are punctuated by these DR sequences. The promoter element seems to be upstream of the array, and initiates transcription of the associated RNA strand that is then cleaved and those spacers are loaded into protein machines that provide specific targeting of the associated invader should it be found. That targeting causes cleavage of the foreign DNA in known cases.

There are three main classes of CRISPR systems; these are type I,II and III, and are established based on the associated protein machines - actually the specific genes that encode for them. In the archaea, only type I and III have been found so far, while in bacteria, all three types have been seen. One of those types, type II, exists in a few species, including *Streptococcus pyogenes*, which encodes for a type II system. The specific gene that is used to classify this system is named Cas9.

We see cases where Cas genes, and sometimes CRISPR arrays, exist in a viral sequence. This, of course, opens many questions about the role of the CRISPR system with respect to viruses - why would a virus carry an immune defense that targets viral sequence? The existence of these arrays suggests that the CRISPR system itself might be mobile. How does the system get established, and how does the surrounding transcription machinery get encoded on the chromosome?

For this assignment, we will work to find the promoter motif. We would expect that this promoter motif and an associated B recognition element (BRE) would be present in archaeal species. What is that motif, and where is it located relative to transcription start?

We know little about the sequence specifics of the promoter element, though it seems to be about 13 nucleotides in length. In Pyrobaculum species, genes are packed quite tightly, so we can look upstream of the array by about 50 bases to see if a common motif sequence might exist. We need to assume that some mutations can be tolerated in that sequence.

Null model

In prior years, we included an option that scrambled the input sequences before running to find a best score. This would effectively eliminate any signal that might be present in the data, while preserving the overall composition of the sequences. We would then compare the best score achieved without scrambling to the score achieved with scrambling. The information that remains in a scrambled set of sequences is only the base level composition of the sequences:

$$Q = Pr_{s \in \{ACTG\}}(s) = \frac{count(s)}{N}$$

We can then use relative entropy to find a score relative to the base level composition by:

$$REscore = \sum_{i=0}^{cols} \sum^{j \in ACGT} P_i(j) \log_2\left(\frac{P_i(j)}{Q(j)}\right)$$

Assignment

Write a BME205 style program (.py file) that is based on the Randomized Motif Search that presented in class/videos and is described in Chapter 2 of the text. Your program should accept a fasta file that contains multiple fasta records as input (STDIN), and your program will then output (STDOUT) the consensus sequence and associated profile score. The score will be the sum of encoding costs (entropies) across each position in the final profile. We will use pseudocounts for this assignment, as described in the text (p. 86-89), and we will need an option to provide the number of pseudocounts (-p).

Produce a notebook that should be fully runable for use during inspections.

I find it easier to accumulate counts rather than calculating a probability based profile. This will also make it a bit easier to deal with pseudocounts when scoring your count-based profile.

$$score = \sum_{i=0}^{cols} \sum^{j \in ACGT} P_i(j) \log_2\left(\frac{P_i(j)}{Q(j)}\right)$$

Randomized Motif Search (p. 93) can be easily trapped in local minima, so we will need to iterate some number of times to find a trajectory that produces the best results. This will need to be an option (-i) that establishes the iteration number.

We also don't know the appropriate motif length that we are looking for, so we need an option (-k) to specify this.

The full command line string should then look something like this:

In [ ]:

```
randomizedMotifSearch.py -i=100000 -p=1 -k=13 <somefile.fa >someOutputFile.fa
```

your program will need to use standard OO style, include docstrings and be well commented. It must run!

I will provide a few fasta files that present the upstream 50 bases from species of the Pyrobaculum group. It is possible that the promotor+BRE that we are looking for is conserved across these species, maybe not perfectly though. Providing more data to a single run may be useful.

I do have expression data for these species, so we know that not all of these arrays are functional - at least under the conditions when I grew them.

Your program should output the final score achieved, along with the associated consensus motif.

Extra Credit

The following optional features would be useful and considered for a combined extra 5 points:

1) -g Use Gibbs sampling to find the optimal consensus motif.

2) -m Print the specific motif and the name of the contributing sequence for each of the participating fasta records.

Each of these options should only be true when the flag is specified by the user (ie. you shouldn't have to explicitly state True/False on the commandline).

Submit your working program to Canvas.

Notes:

1) It is likely that we will need to amend this assignment as we work with it.

2) Sept-2021 eliminated the need for scrambling sequences by using relative entropy

Inspection intro

- What data structure did you use to hold your sequence motifs (DNA)?
- Is your profile structure based on counts or frequencies?
- Where did you implement iterations (mostly why did you choose this)?
- If you implemented Gibbs Sampling, how did you implement recomputing of the profile and motif?

Inspection Results

In [ ]:

```

```