Ben Siranosian
March 4, 2014
CSCI1820 hw4

Problem 0 – Mimivirus

1. Avian Bird Flu virus (Influenzavirus A)
   Mimivirus does not show significant sequence conservation with influenzavirus A.
   Envelope proteins from mimivirus do show some similarity with influenzavirus A, such as "putative glucosamine--fructose-6-phosphate aminotransferase (AAQ09584.2)" which matches the H9N2 sequence with a score of 26.6 and E-value of 0.054.

2. Ebola virus (ebolavirus)
   Mimivirus does not show significant sequence conservation with ebolavirus, and I can't seem to find proteins that are conserved between the two… There is a distinct lack of conservation of mimivirus with other, simpler RNA viruses. Mimi is much more related with bacteria and higher eukaryotes.

3. Mimivirus shows similarity to the bacterium genus Clostridium.
   DNA directed RNA polymerase subunit 1 (AAQ09585.2) matches Clostridium DNA directed RNA polymerase with a bitscore of 176 and E-value of 2e-41.

   Mimivirus also has conserved domains that show up in Clostridium, such as the BclB C-terminal domain.

4. Mimivirus also shows conservation with the more complex organism Saccharomyces cerevisiae (yeast). There is a high level of homology between several proteins discovered using BLAST.

   DNA directed RNA polymerase subunit 1 (AAQ09585.2) matches yeast RNA polymerase II with a bitscore of 740 and E-value of 0.
   DNA directed RNA polymerase subunit 2 (AAQ09583.2) matches yeast DNA directed RNA polymerase with a bitscore of 719 and E-value of 0.

   Mimivirus also shows other domains well conserved with yeast, such as the mRNA capping enzyme, which shows high homology with yeast Abd1p (mRNA capping enzyme, bitscore 80.5, E-value 3e-15) and RNA methyltransferase (bitscore 80.5, E-value 3e-15).
   Clearly, machinery integral for protein expression and mRNA editing is conserved between mimivirus and yeast, showing the evolutionary relationship between them.

Problem 1 – RNAseq

My strategy for this problem is as follows:

a) Record the position of each alignment of each read in the genome.
b) Visualize the number of alignments in a sliding window of length *w* to get an overview of the number and positions of potential genes.
c) Verify each gene with a probabilistic argument.

Our RNAseq dataset has 1878 unique 8-mer reads. There are 4^8=65536 possible 8-mers. Assuming that the genome is random and reads are randomly distributed, the probability a given read aligns at a given

position is simply the probability of picking the read out of the pool of 8-mers, or $p=1/4^8$. We expect to see each RNAseq read $p*2^{18}=4$ times in the genome.

Assuming the null hypothesis that reads are distributed evenly throughout the genome, the probability of a read starting at position *i* can be modeled with a binomial distribution, $B(n=2000, p=1/4^8)$. The mean of this distribution, $np=0.031$, is the expected number of reads to align at a given position.

Within a given window of length *w*, the expected number of reads to align is $w*0.031$. Figure 1 shows my first analysis of a low resolution window of 1000bp. Before looking for genes, it's important to notice that there is a baseline present at about 1000*0.031=31, suggesting that my analysis is correct and regions of the genome without genes conform to the null hypothesis. Potential gene regions are highlighted with a red number. 15 potential regions are identified in this preliminary analysis.

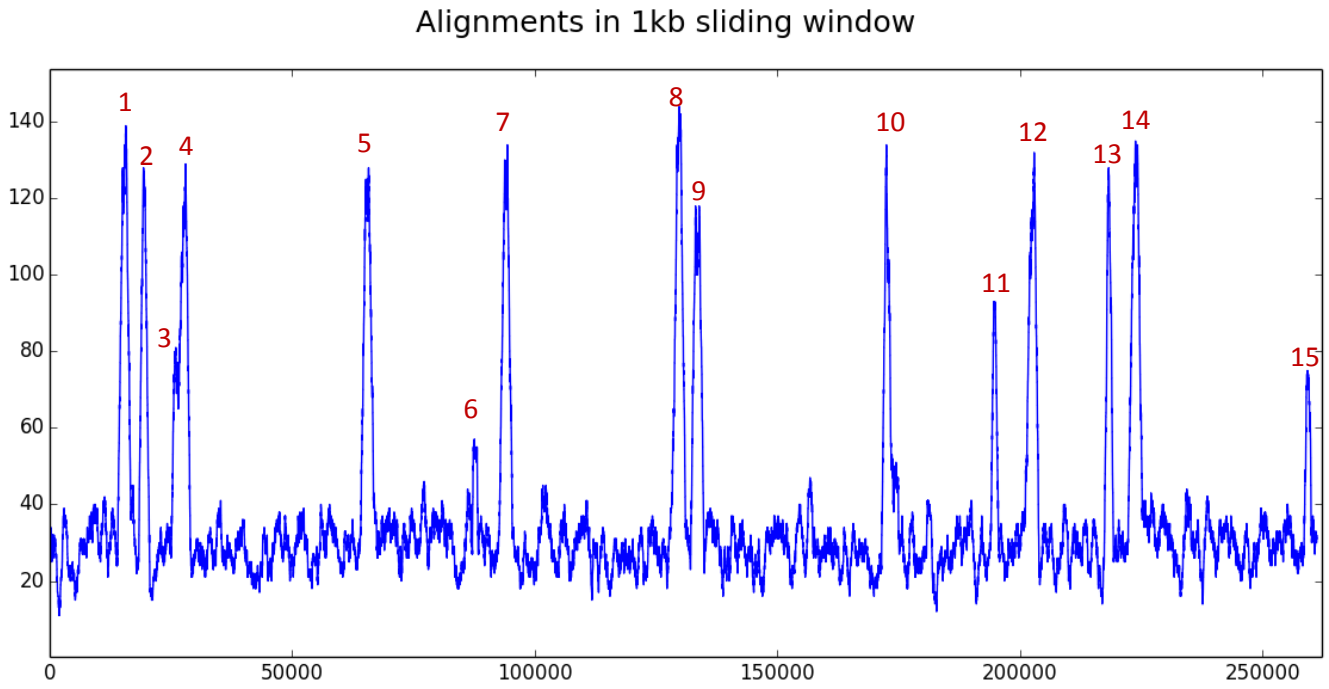


Figure 1

We are not assuming any prior knowledge about gene length or spacing along the genome. To identify genes, I wanted to pick regions that showed significant elevation above the baseline in a sliding window analysis. Although the 1kb window is good for visualization, I used a 500bp window for gene calling to ensure I didn't miss small genes. First, I had to decide on a cutoff value.

Let *X* be the number of reads in a window of length *w*. *p* in this case is the probability of a read from our set aligning to a given position in the genome.

$X \sim B(n=w, p=0.031)$ $E[X]= np = 0.031w$

I wanted a cutoff so that the probability of having *c* or more reads in a window of length *w* was significant. In a window of length 500, the probability of having 35 or more reads is:

Ben Siranosian
March 4, 2014
CSCI1820 hw4

P(success>=35) = 1 – P(success<35)
= 1 – sum k = 0 to 35(P(success=k)) = 1 - 0.99999034 < 10^{-5}

The probability of having 35 or more reads from a window of size 500 under the null model is less than 10^{-5}, a very strict cutoff value that I used for my gene calling. To ensure local variation in sequencing didn't produce a large number of small genes, I allowed the number of alignments to be below the 35 value for 15% of the window length and still consider the position as part of a gene. My results are reported in the table below, where "Alignments" is the number of alignments in the given region, "Expected" is the number of alignments expected in a region of the same size under the null hypothesis and "Probability" is the probability of seeing at least that many alignments in a region of the given size under the null model (calculated using the binomial distribution).
The probability for seeing these gene structures under a null model is extremely low (it's even underflowing on some of the genes), suggesting that they are truly the regions generating the RNAseq reads.

Start	End	Size	Alignments	Expected	Probability
14911	16796	1885	244	58	< 1.11022302463e-16
19224	20691	1467	184	45	< 1.11022302463e-16
26069	26752	683	76	21	< 1.11022302463e-16
27114	29031	1917	227	59	< 1.11022302463e-16
65035	66932	1897	227	59	< 1.11022302463e-16
87993	88389	396	41	12	8.00992605576e-12
93604	95474	1870	223	58	< 1.11022302463e-16
129181	131124	1943	261	60	< 1.11022302463e-16
133089	134903	1814	228	56	< 1.11022302463e-16
172435	173522	1087	143	34	<1.11022302463e-16
173667	174043	376	36	12	1.08638631424e-09
174726	175091	365	34	11	5.85811277265e-09
194937	195703	766	99	24	< 1.11022302463e-16
202069	204000	1931	240	60	< 1.11022302463e-16
218193	219291	1098	143	34	< 1.11022302463e-16
223269	225321	2052	255	64	< 1.11022302463e-16
259555	260099	544	56	17	2.99760216649e-15

Some caveats of my approach: I wanted to identify definite genes without producing too much noise. My approach may miss small "genes" where the number of reads is less than the threshold, or genes that have poor RNAseq coverage. I use a sliding window of size 500bp, which may interpret two genes with a very short distance between them as a single gene. Additionally, to eliminate a large number of small genes being identified from noise, I allow the number of alignments to be below the threshold for no more than 75bp in a single gene. This may concatenate genes with less than 75bp between them, but I found it was necessary to prevent a lot of manual cleanup after the analysis. My threshold value is chosen for this specific example and will need to be modified for a different genome or different number of reads aligning to the genome.

Code for producing these analyses is in RNAseq.py. The script will produce a text output of identified genes as well as a plot of the sliding window analysis with gene calls.

Ben Siranosian
March 4, 2014
CSCI1820 hw4

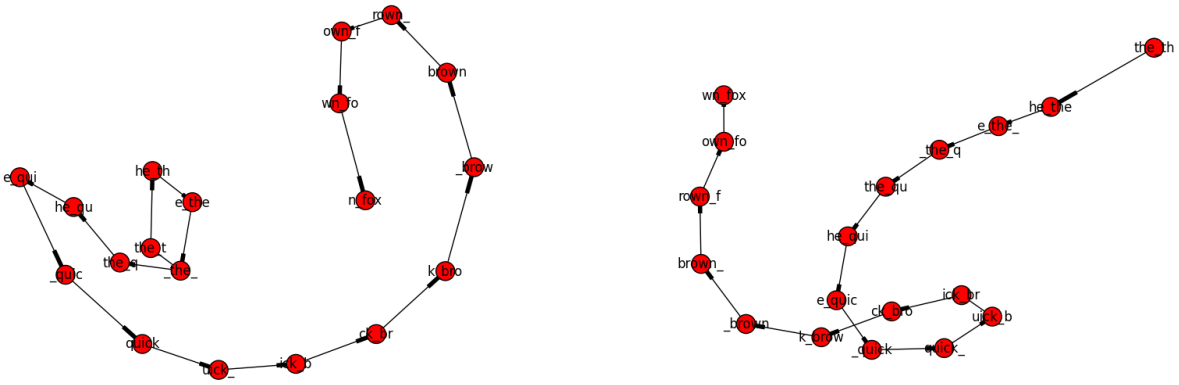
Problem 2 – De Bruijn Graphs

Code for producing De Bruijn graphs can be found in de_brujin.py.

How does k perturb the structure of the graph?
A choice of k is important for resolving repeats in the graph. If there is a repeat of length (k-1) or greater, the graph will have a cycle that can complicate sequence assembly. Increasing k can make it possible to resolve the repeat unambiguously. Consider the following sequence that has a repeat of length 4.

the_the_quick_brown_fox

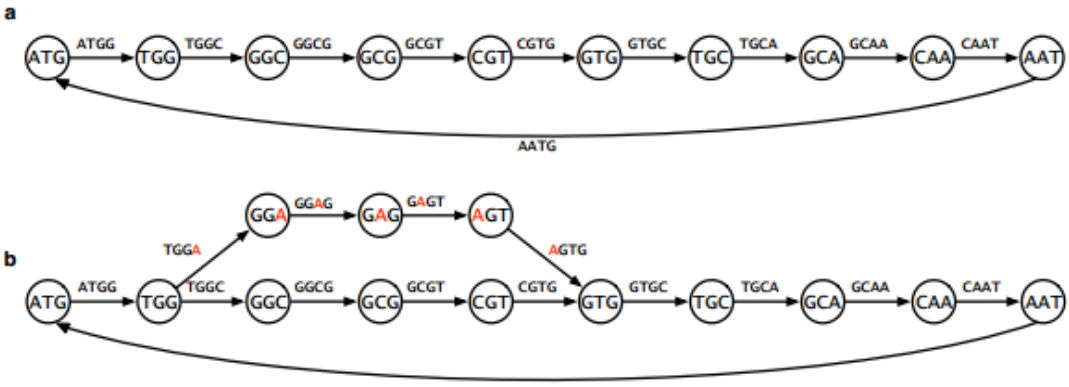
The graph on the left was constructed with k=5. There is a repeat of length 4 in the sequence, though, causing the graph to have a cycle. Compare this to the graph on the right constructed with k=6.



Changing k can have a large effect when trying to assemble sequences as well. I won't present any visualizations because the graphs get too complicated to make sense out of, but k can be thought of as a measure of specificity. Increasing k can make the graph less tangled and more specific. However, sequence can fail to assemble at a higher k when there is not enough overlap between sequnces.

Sequencing errors can dramatically complicate the process of sequence assembly. Each error introduces a "bulge" in the graph above the correct path of assembly. See the figure below (from Compeau, P.E.C., Pevzner, P.A., and Tesler, G. (2011). How to apply de Bruijn graphs to genome assembly. Nat Biotech 29, 987–991) for an example of how introducing a sequencing error in a single read can introduce a bulge. One possible way to resolve bulges is to take the path represented by the greatest number of reads. Sequencing errors are unlikely to effect the same position twice, so the bulge should only have a single path through it, compared to the correct path which should have many reads representing it.

Ben Siranosian
March 4, 2014
CSCI1820 hw4



Problem 3 – information theory

a) Weighing coins

A minimum of 3 trials is needed to determine which coin is lighter. Weigh 4 coins on each side of the scale. If they are the same weight, the coin excluded from the first trial is lighter. Separate the lighter 4 into 2 sets of 2 coins and weigh them. Separate the lighter 2 and weigh them against each other. The lighter coin in this last comparison is the odd one out.

In the general case with *n* coins and one lighter coin, a minimum of floor(log2(n)) comparisons are necessary. At each comparison we halve the number of coins to be compared. In the case of an odd number, two even sets of coins can be compared. If one set is lighter than the other, proceed with that set. If the two sets weigh the same, the coin left out is the light one.

b) 20 questions game

The algorithm for winning is to halve the number of possibilities with each question. In the case with numbers, the first question should be "is your number above $2^n/2$?" Then, proceed with eliminating half of the remaining numbers with each question by asking if her number is higher than the median value of remaining possibilities. In the case with the dictionary, use the same method but ask if Alice's word is after the query word alphabetically.

With the algorithm above it is impossible to detect a lie until all questions have been asked. Bob will also not know when Alice lied: if she lied on the first question, all 19 following it were irrelevant. Therefore, at most, Bob must proceed through the algorithm twice and ask 40 questions before uncovering the true number. To cut down on this upper bound, Bob can ask each question twice in succession. If he gets two different answers, he can ask the question a third time to verify the true response. The number of questions necessary with this method is reduced to (20 + # questions before Alice lies + 1).