BIOL591: Introduction to Bioinformatics
What's wrong with BlastN?

I. Progress in solving the *Mystery of the Missing Match*

I.A. Local *BlastN* vs NCBI's *BlastN*

Tuesday we discovered we have in our hands a working program *BlastN*, written in Perl, that at least in some respects acts like *BlastN* implemented by NCBI . How similar is it? Recall from Tuesday's notes what official BlastN does and examine the program to see if our program does it:

SQ1. Compare official BlastN and BlastN.pl in the following ways:

1. Filter the query sequence to remove repetitive regions. **Does BlastN.pl do this?**
2. Find all matches between the query sequence and the target sequence
   - **a.** Extract a subsequence from the query, called a **word**,… **Does BlastN.pl do this?**
   - **b.** Find an *exact* match of the word in the target sequence. **Does BlastN.pl do this?**
   - **c.** Use a modified Smith-Waterman algorithm to extend the word match in both directions. **Does BlastN.pl do this?**
   - **d.** Calculate a score related to the probability of finding a match as good or better than the final match. **Does BlastN.pl do this?**
   - **e.** Save those matches whose scores are better than a given threshold. **Does BlastN.pl do this?**
   - **f.** Repeat Steps **a** through **e** until there are no more words remaining to try within the query. **Does BlastN.pl do this?**
3. Rank the matches by their scores. **Does BlastN.pl do this?**
4. Print out the top matches. **Does BlastN.pl do this?**

I.B. Mysteries to be solved

There remains the main mystery:

> *Why can't NCBI BlastN (nor ours) find the similarity between RS1140646 and the corresponding region in chimp chromosome 22?*

Some have suggested that perhaps the filtering performed by Blast is the culprit. Maybe Blast filters away the sequence so that there is nothing left to match.

SQ2. Test whether filtering is the key difference in two ways:

- **a. Compare RS1140646 and chimp-rs1140646-region.fa via NCBI's pairwise *Blast*, after disabling filtering. (Look carefully, you *can* do this!) *[check the Scenario if you don't know how to do pairwise Blast]***
- **b. Disable filtering in BlastN.pl and rerun RS1140646 and chimp-rs1140646-region.fa. (How would you do this?)**

I.C. How to solve the ultimate mystery (of *BlastN*, I mean)

Filtering often does make a difference, but in such cases, the sequences usually look highly repetitive, like AAAATAAATA. Not so here. In any case, you've already demonstrated in SQ2 that we need to look beyond filtering.

SQ3. Think seriously on why *BlastN* does not find the match. For starters, why do *you* think it *should* find a match? Make yourself into *BlastN*, and by hand go through all the steps of the *BlastN* algorithm using RS1140646 and the small chimp sequence shown in the Scenario. Do *you* still find a match?

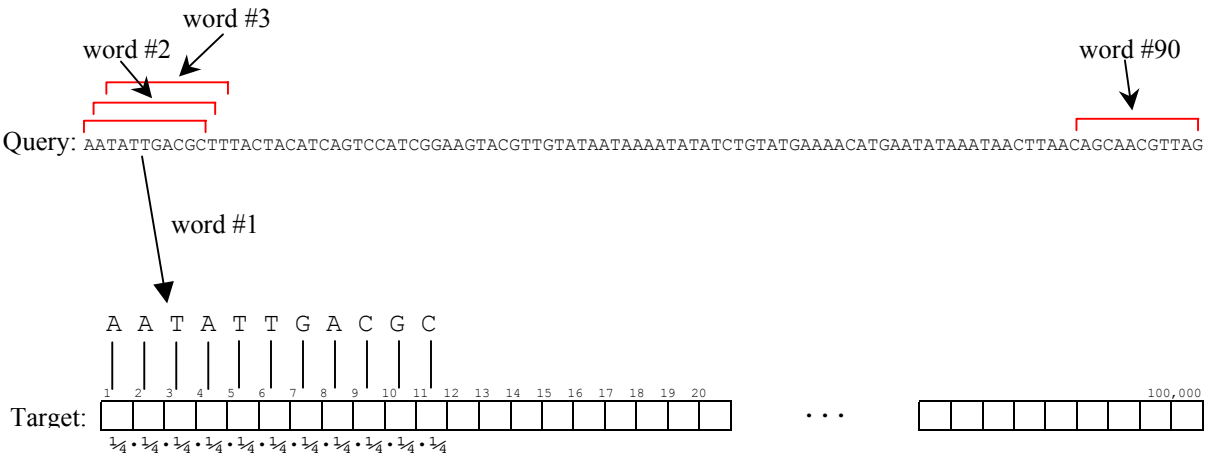
II. Scoring sequence alignments

Our version of *BlastN* prints out every word match it finds, after extending it in both directions as far as it can. Real *BlastN* doesn't do that. Instead, it calculates a score for each match, ranks all matches by their scores, and provides the top **x** matches that exceed a threshold **y** (you can specify **x** and **y**). The score thus serves to order the matches, so you can look at the best ones first, and also tosses out matches that are worse than you want to consider.

What kind of scoring system would be appropriate for sequence alignments? The first idea that might come to mind would be to use directly the raw score that *BlastN* calculates, i.e. one of the numbers found in scoring tables. This is OK for ranking matches, but suppose that *all* of the matches were no good, according to your tastes. How could you prevent *BlastN* from providing you with pages of garbage?

A better idea is to transform the raw score into a probability. If a match would occur frequently in a random sequence, you don't want to know about it. If you'd have to go through 10^{180} random sequences to find a match as good as the one *BlastN* is considering, then you want the program to pass that one on to you. So, we need to talk about expected frequency.

Suppose you're following Blast as it compares a 100-nucleotide sequence with a 100,000-nucleotide sequence. The first step is to extract a word from the query (let's say that the word size had been set to 11 nucleotides). What's the probability that this particular 11 nucleotides match the first 11 nucleotides of the 100,000-nucleotide sequence? Not much, sure, but we need a number. Let's simplify the situation and say that the probability of a match of one nucleotide is ¼ . If so, then:



The probability of a match between word #1 and positions 1-11 of the target is $(¼)^{11}$. But there is the same probability of a match between word #1 and positions 2–12 and 3–13, etc., all the way up to positions 99,990 – 100,000. That's about 99,990 possible matches, each with a probability of $(¼)^{11}$, giving a combined probability of $99{,}990 \cdot (¼)^{11}$. But that's not the end: there are also words #2, #3, all the way up to word #90,[1] so there are now $90 \cdot 99{,}990$ ways of getting a match, each with a probability of $(¼)^{11}$. That gives a total expected number of matches of:

$$90 \cdot 99{,}990 \cdot (¼)^{11}$$

or more generally:

$$(1)\quad \mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot \mathbf{p}^{\mathbf{S}}$$

where **E** is the expected number of matches, **m** is the effective length of the query, **n** is the effective length of the target, **p** is the probability of a single match, and **S** is (for the moment) the total number of matches.

Mathematically, it is more convenient to transform this expression into an exponential based on *e*, forcing the expression to look like:

$$(2)\quad \mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$$

To fit this form, it works out that the constant **λ** has to equal –ln(**p**).[2]

We now have almost what we want: a score that tells us how frequent we should expect a match to be. Unfortunately for this expression, *BlastN* finds matches that are more complicated than a string of matching nucleotides. This means that the expression has to be modified. I don't understand half the math necessary to tell you how the ultimate expression arises, but you can see that this expression used by *BlastN* to calculate expected frequency is pretty close to the expression we derived intuitively from first principles. In the actual expression:

$$(3)\quad \mathbf{E} = \mathbf{K} \cdot \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$$

where **K** and **λ** are empirically derived constants, the expected frequency, **E**, depends on the effective lengths of the query and the target, and it is related exponentially to the score **S**. Furthermore, **λ** is generally close to ln(¼), and **K** (a constant that depends on the base composition of the target) is not far from 1.

SQ4. Let's see these symbols in action. Go to the NCBI web site (see Archives, Bioinformatics Resources for link) and do a *BlastN* search of the chimp rs1140646-region against all available nucleotide sequences. Look at the very end of the results you obtain to find values for K, λ, m, and n. Then go to any match in the results (you'll have to scroll down from the top past the summary list) that has an E-value greater than zero. You'll find the E-value and the raw score, S, in the line that begins:

Score = nnn bits ([raw score]), Expect = [E-value]

Determine from these values whether the given E-value can be calculated by equation (3).

---

[1] Why not 100 words? Because the last word ends at position 100, so must begin at position 90. The number of words = length – wordsize + 1.

[2] $\mathbf{m} \cdot \mathbf{n} \cdot \mathbf{p}^{\mathbf{S}} = \mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$, leads to $\mathbf{p}^{\mathbf{S}} = e^{-\lambda \mathbf{S}}$, and (taking the log of both sides) $\ln(\mathbf{p}) = -\lambda$.

You'll notice in the same line that gave you raw score and **E**-value, there's a quantity called *bits*. This is yet another way of representing the score. One problem with the raw score is that its significance depends on other parameters (**K** and **λ**) that can change depending on the base composition of the sequence being examined. One way to transform the raw score into a score whose significance is independent of these quantities is to force the equation for **E**-value into the form:

$$(4)\ \mathbf{E} = \mathbf{K} \cdot \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$$

$$(5)\quad = \quad \mathbf{m} \cdot \mathbf{n} \cdot 2^{-\mathbf{S}'}$$

leading to:

$$(6)\ \mathbf{S}' = [\ \lambda \cdot \mathbf{S} - \ln(\mathbf{K})\ ] \ / \ln(2)$$

where **S'** represents the normalized score. Since the **E**-value depends in the latter expression only on **S**' and on readily determined quantities (i.e., the effective length of the query sequence, **m**, and the effective length of the target sequence,**n**), it is more transportable. Secondly, since **S**' is present as an exponent of 2, it relates the number of binary digits, or bits, necessary to represent a number, just as the exponent in 10^n tells you how many digits are in the decimal representation of a number. Bits is also the currency of information theory, making it a theoretically attractive unit, one that is interconvertible with the **E**-value. You might also think of bits as the number of binary digits it takes to represent the probability of a specific match as good as the one reported. Multiply this probability by the sizes of the two sequences considered and you get, as in equation (1), the expected number of matches.

SQ5. Using the same *BlastN* result from SQ4, calculate the E-value from the bit score (S') and equation (5), and determine whether this value corresponds well with the given E-value.

Certainly one major difference is that our *BlastN* does not calculate E-values. Consider how you would go about doing this – what you need to know and where in the program you need to know it.

SQ6. Add a line to the program that calculates the E-value for a match and then prints it. To do this, you will need to know (amongst other things) values for lambda and K. Grab those values from the output from a Blast search you do at NCBI. For now, don't bother saving the E-values or think about ranking matches based on E-values.

III. Are sequences nearby CG's more prone to mutation?

One might consider the worries about BlastN to be a technical digression. The ***biological*** bottom line of this research simulation ought to have something to do with an ultimate mechanism of genetic variation in humans. Let's see how even in our embryonic state as bioinformaticians we can reach the edge of human knowledge and go beyond, so long as we're willing to build our own tools.

Today's web site provides you with sequences of every known SNP in human chromosome 22, plus the sequence of human chromosome 22. The SNP sequences are given with flanking sequence, up to 300 nucleotides on each side. If the presence of CG-dinucleotides led to a higher mutation rate, not only at the CG site itself but nearby as well, then we should be able to detect

this by measuring the distance between SNPs and their nearest CG sites. Is this distance random? That is, do SNPs occur in the chromosome without regard to the positions of CG-dinucleotides? Or is the distance nonrandom? Perhaps the frequency of SNPs increases relative to random expectation as one considers nucleotides closer to CG-dinucleotides.

Consider the program Analyze_SNPs.pl, which you can download from today's web site. Much of the program is there, except for the heart of it: How does the program capture the interval between an SNP and its closest CG-dinucleotide? I suggest regular expressions.

SQ7. Make a go of it: Try to complete Analyze_SNPs.pl so that it finds the distance of each SNP to its closest CG-dinucleotide and counts the number of times that distance occurs amongst all SNPs. You should end up with a table looking like this:

Distance	**Count**	
1	***(number of SNPs next to a CG)***	***(example: AACGRTC)***
2	***(number of SNPs 2 nucleotides away)***	***(example: ACGARTC)***
3	***(number of SNPs 3 nucleotides away)***	***(example: AARTCCG)***
. . .		

SQ8. There is a serious problem in using regular expressions that you might encounter – and many ways of overcoming it. If you find this problem, think about a way within your current knowledge to solve the problem but also think of a fix to Perl you would like to see. (Maybe it already exists!)

SQ9. Think carefully about how best to interpret the numbers the final program will produce. Do you need other information to interpret them? If so, how would you get this information?