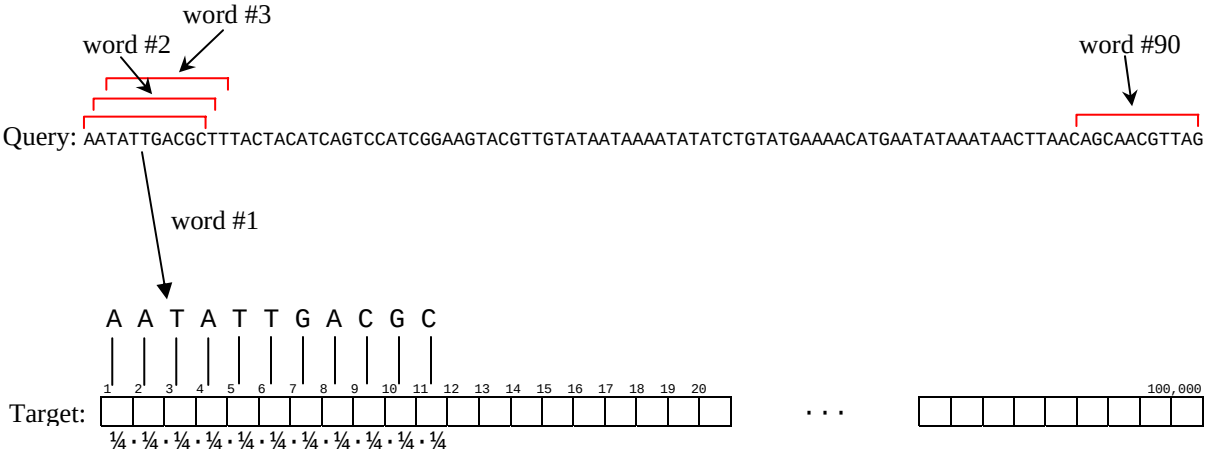
The Truth About Blast
Scoring

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,[a] 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:

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

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

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:

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

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.

SQ5. Let's see these symbols in action. Go to the NCBI web site and do a *BlastN* search of the *lef* toxin sequence M29081 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).

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

---

[a] 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.

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

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

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.

SQ6. Using the same *BlastN* result from SQ1, calculate the E-value from the bit score (S') and equation (5), and determine whether this value corresponds well with the given E-value.

Enough theory! You never learn really something until you try to teach it to someone else. Here's your opportunity. Teach our version of *BlastN* how to calculate E-values.