MST::SeqTools::rSearch()

Gevorg Grigoryan

February 22, 2018

Abstract

Relevant logic and math for the randomized algorithm implemented in MST::SeqTools::rSearch().

1 Problem

Suppose we have a set A of N sequences, each of length L. For each sequence σ, we would like to identify all other sequences in the set that are within some fractional identity c to σ. For example, c could be 0.5 (i.e., 50%), such that we would be looking for all sequences, which when aligned onto σ without gaps would have residues in at least half of the positions match up exactly with σ.

2 Solution

We define a word of a sequence to be a sub-set of its L positions along with the residues at these positions. Two sequences share a word (have a word in common) if they have the same exact residues at the corresponding positions of the word. For a word of length w from σ, emergent from some random subset of w positions, what is the probability that a sequence randomly chosen from the full set would share the same word? Assuming that residues in randomly-chosen positions are idenpendent, this is the product of probabilities of each individual residue matching up. If f_a is the frequency of amino-acid a in the full sequence set, then the latter can be apprximated as:

$$p_e = \sum_{a=1}^{a=20} f_a \cdot f_a \tag{1}$$

Thus, the probability of a randomly-chosen sequence sharing the word is p_e^w and the expected number of such sequences in the full set is $N_w = p_e^w \cdot (N-1)$.

If the desired fractional identity cutoff is c, then the minimum number of identities we are looking for between any two matching sequences is $s = \text{ceil}(c \cdot L)$. If we choose some word W from σ of length no larger than this number (i.e., $w \leq s$), what is the probability that a given match to σ will be found among the set of sequences that share W with σ (lets call this sub-set of sequences Ω_W)? The worst-case scenario (in terms of the lowest probability, on average) would be when the match just scrapes by the identity cutoff, meaning that it has exactly s identities. Then, the probability of finding this match within Ω_W is simply the probability that the w positions we chose for word W are among the s positions that are identical between σ and the match. This probability can be expressed as the ratio of the number of ways to choose w out of s to the number of ways to choose w out of L:

$$p = \frac{C_s^w}{C_L^w} = \frac{s!}{w!(s-w)!}\frac{w!(L-w)!}{L!} = \frac{s!(L-w)!}{(s-w)!L!} = \prod_{k=0}^{w-1}\left(\frac{s-k}{L-k}\right) \qquad (2)$$

So, if we look through all of the N_w sequence in Ω_W, we will (on average) recover p fraction of all matches. But what if we do this n times (choosing a random w-long word each time), and keep appending new matches we discover? When will we come close to finding all matches? To answer this, lets estimate the probability that some match would escape detection after n trials. Since the trials are indepedent, this probability is $(1-p)^n$, and so the fraction of matches recovered after n trials is $1-(1-p)^n$. Suppose we want to reach some desired level of match coverage α (e.g., $\alpha = 0.99$ would give nearly perfect coverage). Then, n needs to be such that $\alpha = 1-(1-p)^n$, or:

$$n = \frac{\log(1-\alpha)}{\log(1-p)} \qquad (3)$$

Note that all of the above is true no matter what word length we choose, subject to $w \leq s$. But the choice of word size may change the cost of the computation. Lets very roughly estimate this cost. Each cycle, we need to explicitly compare σ with on average $N_w = p_e^w \cdot (N-1)$ other sequences. So the total number of sequence comparisons will be:

$$C = n \cdot p_e^w(N-1) = \frac{\log(1-\alpha)}{\log(1-p)}p_e^w(N-1) \qquad (4)$$

In addition to this, there is also a computational cost associated with finding the N_w sequences that share a given word W with σ. This could be implemented in a variety of ways, but lets suppose there is some amortized constant const d per iteration associated with generating the N_w sequences for each σ (we will likely be finding this for all $\sigma \in A$ in one go). The the total cost becomes:

$$\text{cost} = \frac{\log(1-\alpha)}{\log\left(1-\prod_{k=0}^{w-1}\left(\frac{s-k}{L-k}\right)\right)}\left[d + p_e^w(N-1)\right] \tag{5}$$

where we explicitly wrote out all dependence on w. This experession then allows us to optimize word length w to minimize total computational cost, while at the same time ensuring (on average) the desired level of match coverage α.