CS 466 – Introduction to Bioinformatics – Lecture 5

Mohammed El-Kebir

October 4, 2018

Document history:

- 9/17/2018: Initial version.
- 9/17/2018: Fixed incorrect derivation in Section 3.
- 9/17/2018: Additional change in Section 3.
- 9/26/2018: Section 3.
- 10/4/2018: Section 1, to compute suffix(i) longest path from (m, n) to $(i, n/2)$ in reversed edit graph must be computed.

Contents

1 Space Efficient Alignment

We consider the global alignment problem, where we are given two strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and a scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$. The task is to find an alignment of $\mathbf{v}$ and $\mathbf{w}$ with maximum global alignment score among *all* alignments of $\mathbf{v}$ and $\mathbf{w}$. Let $s[i, j]$ (where $i \in \{0, \ldots, m\}$ and $j \in \{0, \ldots, n\}$) denote the maximum global alignment score of the prefix $v_1, \ldots, v_i$ of $\mathbf{v}$ and prefix $w_1, \ldots, w_j$ of $\mathbf{w}$. We define $s[0, 0] = 0$. This leads to the following recurrence:

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases} \tag{1}$$

Recall that a global alignment is a path from $(0,0)$ to (m,n) in the edit graph, which is obtained directly from the above recurrence (e.g., vertices are entries (i,j) and edges correspond to each of the three cases). The maximum score global alignment is the maximum weight path from $(0,0)$ to (m,n). Assume that n is even. Now, suppose that the maximum weight path from $(0,0)$ to (m,n) passes through $(i^*, n/2)$. How can we determine i^*?

We define $\text{wt}(i)$ to be the weight of the maximum weight path from $(0,0)$ to (m,n) passing through $(i, n/2)$. Hence,

$$i^* = \arg\max_{0 \le i \le m}(\text{wt}(i)). \tag{2}$$

How do we compute $\text{wt}(i)$? To that end, we define two functions $\text{prefix}(i)$ and $\text{suffix}(i)$ that denote the weight of the maximum weight path from $(0,0)$ to $(i, n/2)$ and from $(i, n/2)$ to (m,n), respectively. We have that $\text{wt}(i) = \text{prefix}(i) + \text{suffix}(i)$. Now $\{\text{prefix}(0), \ldots, \text{prefix}(m)\}$ can be computed in $O(m)$ space by computing the longest path from $(0,0)$ to $(m, n/2)$, by only keeping two columns in memory starting from column 0. The computed value in $(i, n/2)$ in the last column will correspond to $\text{prefix}(i)$ for each $i \in \{0, \ldots, m\}$. Similarly, $\{\text{suffix}(0), \ldots, \text{suffix}(m)\}$ can be computed in $O(m)$ space by computing the longest path from (m,n) to $(0, n/2)$, by reversing the edges in the edit graph and only keeping two columns in memory starting from column m.

Observe that the running time is the sum of the area of the left rectangle defined by $[(0,0), (m, n/2)]$ and the area of the right rectangle defined by $[(0, n/2), (m,n)]$. Thus, it requires $O(1/2mn) = O(mn)$ time and $O(m)$ space. This will give us $(i^*, n/2)$. How do we get the other vertices of the alignment? We use optimal substructure and simply recurse on the rectangle $[(0,0), (i^*, n/2)]$ to identify vertex $(i^{**}, n/4)$ through which the optimal alignment from $(0,0)$ to $(i^*, n/2)$ has to pass. We also recurse on the rectangle $[(i^*, n/2), (m,n)]$ to identify vertex $(i^{***}, 3n/4)$ through which the optimal alignment from $(i^*, n/2)$ to (m,n) has to pass. The recursion terminates when the rectangle only has two columns.

It is clear that only the initial call of this procedure requires $O(m)$ space, subsequent calls require less space. As described previously, the running time of a recursive call is given by the input area. Now in each subsequent recursive call the area is halved. Thus, the running time is given by $\text{area} + \text{area}/2 + \text{area}/4 + \ldots$. This is a geometric series, and is bounded by $2 \cdot \text{area}$. Thus, the running time is $2(m+1)(n+1) = O(mn)$.

2 Subquadratic Time Alignment

We consider the block alignment problem, where we are given where we are given two strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and block length t. The task is to find a *block alignment* of $\mathbf{v}$ and $\mathbf{w}$. In such an alignment every block in one sequence is either aligned against an entire block in the other sequence or inserted/deleted as a whole. In other words, the alignment path must enter and leave a block through its corner vertices.

We will use the Four Russians technique. Assuming that $\mathbf{v}$ and $\mathbf{w}$ are sequences from a four-letter alphabet (such as DNA), we set $t = \log_2(n)/4$. The algorithm is very simple. We will precompute all alignments of all pairs of strings, each of length $t = \log_2(n)/4$. How

many four-letter strings are there of such length?

$$\begin{aligned} 4^t &= 4^{\log_2(n)/4} && (3)\\ &= (2^2)^{\log_2(n)/4} && (4)\\ &= 2^{\log_2(n)/2} && (5)\\ &= (2^{\log_2(n)})^{1/2} && (6)\\ &= n^{1/2} = \sqrt{n}. && (7) \end{aligned}$$

Each alignment that we want to precompute is between a pair of four-letter strings of length t. Thus, the number of alignments that we need to precompute is $n^{1/2} \cdot n^{1/2} = n$. Now each alignment of takes $O(t^2) = O((\log_2 n)^2)$ time. Thus, the total time needed for all computing all alignments is

$$O(nt^2) = O(n(\log_2 n)^2) = O(n \log^2 n). \tag{8}$$

The precomputed alignments are stored in a look-up table S, indexed by two four-letter strings of length t. We assume that each lookup takes $O(\log n)$ time.

Now let's focus on computing the block alignment between the input strings $\mathbf{v}$ and $\mathbf{w}$. For simplicity, we assume that $|\mathbf{v}| = |\mathbf{w}| = n$. As discussed in class, computing the optimal block alignment requires $\frac{n}{t} \times \frac{n}{t}$ lookups in S, each lookup requiring $O(\log n)$ time. Thus, we require

$$O(\frac{n}{\log_2(n)/4} \cdot \frac{n}{\log_2(n)/4} \cdot \log n) = O(\frac{n^2}{\log n}) \tag{9}$$

time to compute the optimal block alignment given the lookup table S. This running time dominates the time needed to precompute S (which took $O(n \log^2 n)$ time). Hence, we have a total running time of $O(n^2/\log n)$.

3 Is $n^2/\log n$ in $O(n^{2-\epsilon})$ for some $\epsilon > 0$?

Let's assume that this is the case and let $b > 0$ be the base of the logarithm. That means that there exist constants $n_0, c > 0$ and $\epsilon \in (0, 2)$ such that

$$\frac{n^2}{\log_b n} \leq cn^{2-\epsilon} \qquad \forall n > n_0. \tag{10}$$

To verify whether such constants exist, we consider the limit of the ratio between the functions $n^2/\log_b n$ and $n^{2-\epsilon}$. Per our assumption, we expect the limit to be a constant (more specifically, a function of ϵ). We have,

$$\lim_{n\to\infty} \frac{n^2/\log_b n}{n^{2-\epsilon}} = \lim_{n\to\infty} \frac{n^2 n^\epsilon}{n^2 \log_b n} = \lim_{n\to\infty} \frac{n^\epsilon}{\log_b n}. \tag{11}$$

To compute this limit, we use l'Hôpital's rule. Let $f(n) = n^\epsilon$ and $g(n) = \log_b n$. We have $f'(n) = \epsilon n^{\epsilon-1}$ and $g'(n) = 1/(n \ln b)$. As $\lim_{n\to\infty} f(n) = n^\epsilon = \infty$, $\lim_{n\to\infty} g(n) = \log_b n = \infty$

and $g'(n) \neq 0$ for all $n > 0$ (recall that $b > 0$), we meet the preconditions of the rule. Thus, we have

$$\lim_{n\to\infty} \frac{\epsilon n^{\epsilon-1}}{1/(n \ln b)} = \lim_{n\to\infty} \epsilon n^{\epsilon-1} n \ln b = \epsilon \ln b \lim_{n\to\infty} n^{\epsilon} = \infty. \tag{12}$$

This yields a contradiction. Hence, there is no constant $\epsilon > 0$ such that $n^2/\log n \in O(n^{2-\epsilon})$.