CS 466 – Introduction to Bioinformatics
Lecture 9

Mohammed El-Kebir

September 23, 2020

Document history:

- 10/3/2018: Initial version
- 10/17/2018: Fixed incorrect running time of tree alignment
- 10/17/2018: Typos
- 10/25/2018: Typos
- 9/20/2019: Removed "Carrillo-Lipman" section, fixed various typos in proofs
- 9/18/2020: Several edits
- 9/23/2020: Typo in Equation 4, now stating that δ is a distance function in Theorem 2

Contents

1 Problem Statement

Let Σ be the alphabet. We are given k strings $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$. A *multiple alignment* $A = [a_{p,i}]$ is defied as an $k \times \ell$ matrix where $\ell \in \{\max_{p \in [k]}\{|\mathbf{v}_p\}, \ldots, \sum_{p=1}^{k} |\mathbf{v}_p|\}$ such that (i) each entry $a_{p,i}$ is a character from the gap-extended alphabet $\Sigma \cup \{-\}$, (ii) removal of the gap characters from each row $\mathbf{a}_p$ yields input string $\mathbf{v}_p$ and (iii) there is no column $j \in [\ell]$ consisting of only gap characters in A, i.e. $a_{p,j} = -$ for all $p \in [k]$.

We consider the *Sum-of-Pairs (SP) score* $\mathrm{SP}(A)$, which uses a given pairwise scoring function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ to score every column of an alignment A by considering all pairs of input sequences. Specifically, $\mathrm{SP}(A)$ is defined as

$$\mathrm{SP}(A) = \sum_{p=1}^{k} \sum_{q=p+1}^{k} \sum_{i=1}^{\ell} \delta(a_{p,i}, a_{q,i}). \tag{1}$$

We have the following two problems.

Problem 1. WEIGHTED SP-EDIT DISTANCE *Given strings* $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$ *and a scoring function* $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$*, find a multiple alignment* A *such that* $\mathrm{SP}(A)$ *is minimum.*

Problem 2. SP-GLOBAL ALIGNMENT *Given strings* $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$ *and a scoring function* $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$*, find a multiple alignment* A *such that* $\mathrm{SP}(A)$ *is maximum.*

Observe that the two problems differ only in the direction of their objective functions, minimization vs. maximization.

2 Tree and Star Alignments

In general, heuristics come with no hard, theoretical guarantees on their worst-case performance. For instance, the greedy progressive alignment algorithm that we saw in class has no such guarantees. In other words, we do not know how far off the cost of the returned solution is from the optimal cost. In this section, we will describe a constant-factor approximation algorithm that comes with theoretical guarantees on its performance. That is, the cost of a returned solution is at most a constant factor more than the optimal cost.

Let $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$ be our input strings. Recall that $D(\mathbf{v}_i, \mathbf{v}_j)$ is the optimal (weighted) edit distance between $\mathbf{v}_i$, and $\mathbf{v}_j$. We start with the following key definition.

Definition 1. *Let* T *be a tree with* k *nodes, where each node is labeled with a distinct string from* $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$*. Then, a multiple alignment* A *of* $\mathbf{v}_1, \ldots, \mathbf{v}_k$ *is called* consistent *with* T *if the induced pairwise alignment of* $\mathbf{v}_i$*, and* $\mathbf{v}_j$ *has cost* $D(\mathbf{v}_i, \mathbf{v}_j)$ *for each edge* $(\mathbf{v}_i, \mathbf{v}_j)$ *of* T*.*

The following theorem states that it is easy to compute an alignment that is consistent with a given tree T.

Theorem 1 (Gusfield [1])**.** *Let* T *be a tree whose* k *nodes are each labeled by a distinct string from* $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$*. We can compute an alignment* $A(T)$ *of* $\mathbf{v}_1, \ldots, \mathbf{v}_k$ *that is consistent with* T *in* $O(k^2 n^2)$ *time.*

Proof. Without loss of generality, we assume that the vertices have been arranged such that $\{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$ induces a connected subtree T' of T for each $i \in [k]$. We will show that the theorem holds, by proving inductively that adding each string $\mathbf{v}_i$ while maintaining consistency takes $O(in^2)$ time.

The base case $i = 2$ is trivial, amounting to a pairwise alignment of $\mathbf{v}_1$ and $\mathbf{v}_2$ that by definition is consistent with tree T that connects the two vertices by a single edge. Computing the optimal pairwise alignment takes $O(in^2) = O(2n^2) = O(n^2)$ time.

As for the step $i > 2$, by the induction hypothesis we are given a tree T' that is consistent with strings $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i-1}$. Let $\mathbf{v}_i$ be a string adjacent in T' to one of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i-1}$. Let $\mathbf{v}_j$ be the vertex that is adjacent to $\mathbf{v}_i$ in T'. Let $\bar{\mathbf{v}}_j$ denote the gapped sequence corresponding to $\mathbf{v}_j$ in the multiple alignment $A(T')$. We align $\bar{\mathbf{v}}_j$ and $\mathbf{v}_i$ with the added rule that $\delta(-, -) = 0$. That is, two opposing gaps have a cost of 0.

Let $\bar{\mathbf{v}}'_j$ and $\bar{\mathbf{v}}_i$ be the two resulting gapped sequences. If the optimal alignment does not insert any new gaps into $\bar{\mathbf{v}}_j$ then we add $\bar{\mathbf{v}}_i$ to $A(T')$. The result is a multiple alignment with one more string, where the induced cost of $\bar{\mathbf{v}}'_j$ and $\bar{\mathbf{v}}_i$ equals $D(\mathbf{v}_j, \mathbf{v}_i)$ and where the induced costs from the previous alignment remain unchanged. However, if the optimal alignment inserted a new gap into $\bar{\mathbf{v}}_j$ between characters l and $l+1$, we insert a gap between columns l and $l+1$ in each sequence of the original multiple alignment $A(T')$. Observe that the induced costs of the original alignment remain unchanged, whereas the induced alignment of $\bar{\mathbf{v}}'_j$ and $\bar{\mathbf{v}}_i$ has cost $D(\mathbf{v}_j, \mathbf{v}_i)$. Thus, the new alignment is consistent with the tree T' extended by the edge $(\mathbf{v}_j, \mathbf{v}_i)$. As for the running time, observe that the given alignment $A(T')$ composed of $(i - 1)$ sequences has a length of at most $(i - 1)n$ (recall that length of pairwise alignment of two sequences of length m and n is at most $m + n$). Thus, worst case, $\bar{\mathbf{v}}_j$ has length $(i - 1)n = O(in)$ while $\mathbf{v}_i$ has length n. Thus, it takes $O(in^2)$ time to compute $\bar{\mathbf{v}}'_j$ and $\bar{\mathbf{v}}_i$.

The total running time is $\sum_{i=1}^{k-1} O(in^2) = O(k^2n^2)$. Hence, we can compute an alignment $A(T)$ of $\mathbf{v}_1, \ldots, \mathbf{v}_k$ that is consistent with T in $O(k^2n^2)$ time. □

2.1 Star Alignment

We begin with the following definition.

Definition 2. *A cost function $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ satisfies the* triangle inequality *if*

$$\delta(x, z) \le \delta(x, y) + \delta(y, z) \tag{2}$$

for all $x, y, z \in \Sigma \cup \{-\}$.

Recall that $D(\mathbf{v}_i, \mathbf{v}_j)$ is the optimal (weighted) edit distance between $\mathbf{v}_i$ and $\mathbf{v}_j$. We have the following definition.

Definition 3. *Given strings $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \Sigma^*$, the* center string *$\mathbf{v}_c$ (where $c \in [k]$) is the input string that minimizes $\sum_{i=1}^{k} D(\mathbf{v}_c, \mathbf{v}_i)$. The* center star *is a star tree of k nodes with the center node labeled by $\mathbf{v}_c$ and each of the remaining $k - 1$ nodes labeled by a distinct string from $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \setminus \{\mathbf{v}_c\}$.*

We use Theorem 1 to obtain an alignment A_c of $\mathbf{v}_1, \ldots, \mathbf{v}_k$ consistent with the center star. Let $d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c))$ denote the pairwise alignment cost of $\mathbf{v}_i$ and $\mathbf{v}_j$ induced by A_c. Clearly, $d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c)) \ge D(\mathbf{v}_i, \mathbf{v}_j)$. We introduce the shorthand $d(A_c) = \sum_{i<j} d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c))$.

Lemma 1. *Let δ be a cost function that satisfies the triangle inequality and let $\mathbf{v}_c$ be the center string of a center star alignment A_c of input strings $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Then, for any input string $\mathbf{v}_i$ and $\mathbf{v}_j$, it holds that*

$$d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c)) \leq d(\mathbf{v}_i(A_c), \mathbf{v}_c(A_c)) + d(\mathbf{v}_c(A_c), \mathbf{v}_j(A_c)) = D(\mathbf{v}_i, \mathbf{v}_c) + D(\mathbf{v}_c, \mathbf{v}_j). \tag{3}$$

Proof. Consider any column of A_c. Let x, y and z be the characters in this column of gapped sequences $\mathbf{v}_i(A_c)$, $\mathbf{v}_c(A_c)$ and $\mathbf{v}_j(A_c)$, respectively. By the triangle inequality, we have that $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$. Thus, $d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c)) \leq d(\mathbf{v}_i(A_c), \mathbf{v}_c(A_c)) + d(\mathbf{v}_c(A_c), \mathbf{v}_j(A_c))$. By definition of A_c, it follows that $d(\mathbf{v}_i(A_c), \mathbf{v}_c(A_c)) + d(\mathbf{v}_c(A_c), \mathbf{v}_j(A_c)) = D(\mathbf{v}_i, \mathbf{v}_c) + D(\mathbf{v}_c, \mathbf{v}_j)$. □

We are now ready to state our theorem. Let $\delta : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$ be a distance function, thus satisfying (i) identity of indiscernibles (i.e. $\delta(x, x) = 0$ if and only if $x = x$[1], (ii) symmetry ($\delta(x, y) = \delta(y, x)$) and (iii) the triangle inequality (Definition 2). Let A^* be an optimal alignment of strings $\mathbf{v}_1, \ldots, \mathbf{v}_k$ with cost $d(A^*)$, and let A_c be an alignment consistent with the center star with center string $\mathbf{v}_c$ and cost $d(A_c)$.

Theorem 2. *$d(A_c)/d(A^*) \leq 2(k-1)/k < 2$.*

Proof. We start by defining

$$f(A_c) = \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c)) \text{ and } f(A^*) = \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} d(\mathbf{v}_i(A^*), \mathbf{v}_j(A^*)). \tag{4}$$

Clearly, $2d(A^*) = f(A^*)$ and $2d(A_c) = f(A_c)$. Recalling that $D(\mathbf{v}_i, \mathbf{v}_j)$ is the optimal (weighted) edit distance between $\mathbf{v}_i$ and $\mathbf{v}_j$, we have by Lemma 1 that

$$\begin{aligned} f(A_c) &= \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} d(\mathbf{v}_i(A_c), \mathbf{v}_j(A_c)) && (5)\\ &\leq \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} [d(\mathbf{v}_i(A_c), \mathbf{v}_c(A_c)) + d(\mathbf{v}_c(A_c), \mathbf{v}_j(A_c))] && (6)\\ &= \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} [D(\mathbf{v}_i, \mathbf{v}_c) + D(\mathbf{v}_c, \mathbf{v}_j)]. && (7) \end{aligned}$$

Observe that for any fixed j, each of the terms $D(\mathbf{v}_c, \mathbf{v}_j)$ and $D(\mathbf{v}_j, \mathbf{v}_c)$ show up $k-1$ times. Furthermore, observe that $D(\mathbf{v}_c, \mathbf{v}_j) = D(\mathbf{v}_j, \mathbf{v}_c)$. Thus, we have

$$f(A_c) \leq \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}}^{k} [D(\mathbf{v}_i, \mathbf{v}_c) + D(\mathbf{v}_c, \mathbf{v}_j)] = 2(k-1)\sum_{j=1}^{k} D(\mathbf{v}_c, \mathbf{v}_j). \tag{8}$$

[1] We only need the reverse direction, i.e. $x = x$ implies $\delta(x, x) = 0$.

From the other side, we have

$$
\begin{aligned}
f(A^*) &= \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} d(\mathbf{v}_i(A^*), \mathbf{v}_j(A^*)) && (9)\\
&\geq \sum_{\substack{(i,j)\in[k]^2:\\ i\neq j}} D(\mathbf{v}_i, \mathbf{v}_j) && (10)\\
&= \sum_{i=1}^{k}\sum_{j=1}^{k} D(\mathbf{v}_i, \mathbf{v}_j) && (11)
\end{aligned}
$$

Now, the crucial observation is that the sum $\sum_{i=1}^{k}\sum_{j=1}^{k} D(\mathbf{v}_i, \mathbf{v}_j)$ of the minimum costs of all ordered pairs of strings equals summing the cost of k different stars, each centered around one of the k input strings. We picked $\mathbf{v}_c$ such that it was the star with smallest cost. Thus, we have

$$
\begin{aligned}
f(A^*) &\geq \sum_{i=1}^{k}\sum_{j=1}^{k} D(\mathbf{v}_i, \mathbf{v}_j) && (12)\\
&\geq k\sum_{j=1}^{k} D(\mathbf{v}_c, \mathbf{v}_j) && (13)
\end{aligned}
$$

Hence, we have

$$
\frac{d(A_c)}{d(A^*)} = \frac{f(A_c)}{f(A^*)} \leq \frac{(2k-1)\sum_{j=1}^{k} D(\mathbf{v}_c, \mathbf{v}_j)}{k\sum_{j=1}^{k} D(\mathbf{v}_c, \mathbf{v}_j)} = \frac{2(k-1)}{k} < 2. \qquad (14)
$$

□

References

[1] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, USA, 1997.