Algorithmic Bioinformatics DD2450, spring 2010, Lecture 4

Lecturer Jens Lagergren
Several current and previous students
will be acknowledged in a separate document.

April 18, 2010

1 Local Alignment

When aligning two sequences using local alignment only smaller parts of the sequences are considered.

1.1 Motivation

Local alignment is preferred over global alignment in certain applications:

- Local alignment of two multidomain protein can find common domains.
- Homologous but distant proteins may have diverged a lot and may only share a motif.
- Upstream regions of non-homologous genes can share regulatory elements, but nothing else.

1.2 Defintion

The term *substring* is used for the smaller parts of the sequences. A substring is simply a subset of the consecutive symbols in a string with the order of the symbols preserved.

Notation 1. *$X' \subseteq X$ means X' is a substring of X.*

Input: Two sequences X and Y.

Output: Optimal local alignment score of X and Y.

The local optimal aligning score, λ, is the maximum global alignment score over all substrings. More formally, the local optimal aligning score of two sequences X and Y, with γ defined as earlier, is:

$$\lambda(X,Y) = \max_{\substack{X' \subseteq X \\ Y' \subseteq Y}} \gamma(X', Y')$$

1.3 Dynamic programming procedure

Definition 1. *An (i,j)-local alignment is a global alignment of $x_{i'},\ldots,x_i$ and $y_{j'},\ldots,y_j$, for some $i' < i$, $j' < j$.*

Let $l(i,j)$ be the maximum score, with linear gap cost, over all (i,j)-local alignments π of X and Y. By again looking at the last column of an alignment π, we derive the recursion for l:

$$l(i,j) = \max \begin{cases} l(i-1,j-1) + s(x_i, y_j) \\ l(i-1,j) - d \\ l(i,j-1) - d \\ 0 \end{cases}$$

The base cases are:

$l(0,0) = 0$.

$l(-1,j) = l(i,-1) = 0$, $\forall i,j$

Unlike the case with global alignings, the optimal local aligning score is not guaranteed to be in a certain element of the computed matrix. Finding the optimal score requires a complete search for the maximum value of l. By using back-pointers, starting from the element with the maximum value, it is also possible to get the alignment corresponding to the optimal score.

Algorithm complexity

Time: $O(nm)$.
A constant amount of work is needed for each matrix element.

Memory usage for the optimal alignment matrix using back-pointers: $O(mn)$.

Memory usage for the optimal score only: $O(min(m,n))$
Only the previous and the current row of l are needed in memory.

2 Global alignment with linear gap cost in linear memory

We will now present an algorithm for optimal alignment with memory complexity $O(n+m)$. Our earlier algorithms for computing alignment score has a memory complexity of $O(min(n,m))$. If we are also interested in getting the optimal alignment the memory requirements has been $\Theta(nm)$. When aligning large DNA-strings, memory problems are more sever than time problems.

2.1 Dynamic programming solution

Input: $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$. (the usual sequences)
Assume for simplicity that m is divisible by 2. We can view an alignment as a path in a a weighted graph. A path from $(0,0)$ to (m,n) in the graph corresponds to an alignment. We can represent an alignment path as a list of ordered pairs:

$$< 0,0 >=< s_1,t_1 >, \dots, < s_p,t_p >=< n,m >$$

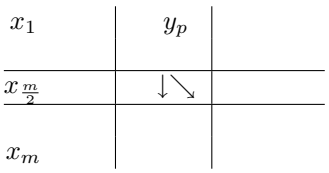
where $s_{i+1} = s_i + \delta$ and $t_{i+1} = t_i + \delta'$ for δ, $\delta' \in \{0,1\}$ and $\delta + \delta' \geq 1$.
We now define the paths passage as the t_i such that $S_i = \frac{m}{2}$. If an alignment path contains several pairs $< \frac{m}{2}, t_i >$, we choose a passage such that t_i has the highest index i.

Observation: There is a 1-to-1 correspondence between alignments and alignment paths.

Algorithm The idea is to use the passage for divide and conquer.
Input: $X = x_1, \dots, x_m$ and $Y = y_1, \dots, y_n$.

1. If $|X| = 1$ or $|Y| = 1$, return an alignment $\pi_1, \dots, \pi_k$ of X and Y (computed in a straigthforward non-recursive way)
2. Compute the passage t for an optimal alignment path p
3. Align $X^{m/2}$ with Y^p, which gives the optimal alignment path $\pi_1, \dots, \pi_k$
4. Align $x_{\frac{m}{2}+1}, \dots, x_m$ with $y_{p+1}, \dots, y_n$, which gives the optimal alignment path $\pi'_1, \dots, \pi'_k$
5. Return the alignment of X and Y: $\pi_1, \dots, \pi_k, \pi\prime_1, \dots, \pi'_k$

A closer look at the passage (an illustration)



We match to symbols whenever we use $\searrow$. So positions in $X^{m/2}$ are only matched to positions in Y^p or blanks. And the other way around.
$\Rightarrow$ An optimal alignment can be obtained by concatenating an optimal alignment of $X^{m/2}$ and Y^p with an optimal alignment of $x_{m/2+1}, \dots, x_m$ and $y_{p+1}, \dots, y_n$.

Computing the passage in time $O(nm)$ and space $O(n+m)$
Recursion for score(s) and passage(p)

$$g(i,j) = \max \begin{cases} g(i-1,j-1) + s(x_i, y_j) \\ g(i-1,j) - d \\ g(i,j-1) - d \end{cases}$$

A passage $p(m,n)$, where $p(i,j)$ is defined as:

$$p(i,j) = \begin{cases} 0 & \text{if } i < \frac{m}{2} \\ j & \text{if } i = \frac{m}{2} \\ \text{if } i > \frac{m}{2} & \\ \quad p(i-1,j-1) & \text{if } g(i,j) = g(i-1,j-1) + s(x_i, y_j) \\ \quad p(i-1,j) & \text{if } g(i,j) = g(i-1,j) - d \\ \quad p(i,j-1) & \text{if } g(i,j) = g(i,j-1) - d \end{cases}$$

Complexity We now analyze the time and space complexity of the entire algorithm.

Memory Computing the passage p can be done in linear memory, because when we compute $g(i,j)$ and $p(i,j)$ on row i we only need the values on the preceding row and the already computed values on the current row. The length of a row is n, which gives memory complexity $O(n)$. We will compute a number of passages ($m-1$ if $m = 2^p$ for any p), but we only compute one passage at the time so we can reuse the memory for each passage computation. Storing the alignment requires $O(n+m)$ memory. Assume inductively that the memory required for a pair of sequences of length n' and m' is bounded by $c(n'+m')$.

Step 1 space $\leq O(n)$

Step 2 space $\leq O(m+n)$ i.e. $c'(m+n)$ for constant c'

Step 3 space $\leq 1 + c(\frac{m}{2} + p)$

Step 4 space $\leq \frac{m}{2} + p + c(\frac{m}{2} + n - p) \leq c(m+n)$
when $c > 1$

Step 5 space $\leq c(m+n)$

So the whole algorithm requires $O(n+m)$.

Analysis of time complexity By induction: Assume that $T(m',n') \leq 2cn'm'$, when $m'+n' < m+n$. Then we get (for $m > 1$)

$$\begin{aligned} T(m,n) &\leq cnm + T(\frac{m}{2}, p) + T(\frac{m}{2}, n-p) \\ &\leq cnm + 2c\frac{m}{2}p + 2c\frac{m}{2}(n-p) \\ &= 2cnm. \end{aligned}$$

Where the second inequality follows our inductive assumption. We conclusion that $T(m,n) \leq 2cnm$. So, the time complexity is $O(nm)$.
Lets compare $(2cnm)$ with the time required for the basic quadratic space algorithm. The basic algorithm takes about the same time as computing the passage (which takes time cnm). The algorithm for computing the global alignment in linear memory thus takes about twice the time of the basic algorithm.

3 Multi alignment

In a multi alignment more than two sequences are compared. This alignment contains more information than pairwise alignments. From the multi alignment it is possible to construct a consensus sequence which can be used to recreate what the ancestor to the original sequences looked like.

Exemple of pairwise alignment:
S_1: ...AATGCG...
S_2: ...ACCGCT...

Exemple of multi alignment:

S_1:	AATGCG
S_2:	ACCGCT
S_3:	AATCCT
Consensus:	AATGCT

Applications:

- Database searches, create HMM
- First step in phylogeny
- Signal identification

Definition: *A multialignment of sequences $S_1, \ldots, S_r$ is a $r \times k$ matrix such that:*

- The support of row i is S_i.
- No column contains only blank symbols.

Let $|A|$ be the number of columns in A and let $s : \{A, C, G, T, -\}^2 \to R$ be a nucleotide similarity function.

Definition: *Sum of Pairs (SP) score, σ, of A is defined as:*

$$\sigma(A) = \sum_{i=1}^{|A|} \sum_{i \leq j < k \leq r} s(A_{ji}, A_{ki})$$

where $|A| =$ *the number of columns in* A.

$$\gamma(S_1, \ldots, S_r) = \max_{\text{align. } A \text{ of } S_1,\ldots,S_r} \sigma(A)$$

Notation: *For a string* $X = x_1, \ldots, x_m$

1. $X^{\leq i} = X_1, \ldots, X_i$
2. $X^{>i} = X_{i+1}, \ldots, X_m$
3. $X^{>i, \leq i+1} = X_i$
4. $X^{>i, \leq i} = -$

Let

$$g(i_1, \ldots, i_r) = \gamma(S_1^{\leq i_1}, \ldots, S_r^{\leq i_r})$$

Recursion for g:

$$g(\vec{0}) = 0$$

$$g(i_1, \ldots, i_r) = -\infty, \ \forall i_1, \ldots, i_r \in \{-1, 0\} \text{ such that } \sum_{j=1}^{r} i_j < 0$$

$$g(i_1, \ldots, i_r) = \min_{\delta_1 \delta_r \in \{0,1\}^r \setminus \vec{0}} g(i_1 - \delta_1, \ldots, i_r - \delta_r) + \sum_{i \leq j < k \leq r} s(S_j^{>i_j - \delta_j \leq i_j}, S_k^{>i_k - \delta_k \leq i_k})$$

Complexity:
Time: $O(2^r n^r)$ since filling an element in the matrix takes time 2^r and the matrix has n^r elements.
Space: $O(n^r)$ for score and alignment.