Dynamic Programming for Bioinformatics...

Global Sequence Alignment

Sequence Alignment. In bioinformatics, **sequence alignment** is the arrangement of two (or more) DNA sequences in either nucleotide or amino acid alphabets that identifies similarity between the sequences that may be a consequence of functional, structural or evolutionary relationships between the sequences[1].

Global Sequence Alignment. Global sequence alignment is the alignment of two (or more) sequences that attempts to align every single position in one sequence against every single position in the other sequence(s).

Global sequence alignment is usually used whenever the sequences involved are of similar length and are expected to be similar in all regions.

General global sequence alignment problem for two sequences. Given two DNA sequences S and T in *either* the nucleotide or amino acid alphabet, determine the best global alignment between them.

Needed to resolve:

- Formal definition of **alignment**.
- Criteria for **best alignmnent** determination.

Formal definition. Let $S = s_1 \dots s_n$ and $T = t_1 \dots t_m$ are two strings (sequences). An alignment $a(S,T)$ is a set of pairs $\{(x_1, y_1), \dots (x_k, y_k)\}$, such that:

1. $k \leq m + n$
2. $x_i = s_j$ for some $1 \leq j \leq n$, or $x_i =$ "_".
3. $y_i = t_j$ for some $1 \leq j \leq m$, or $y_i =$ "_".
4. for a pair (x_i, y_i), either $x_i \neq$ "_" or $y_i \neq$ "_".

[1] http://en.wikipedia.org/wiki/Sequence_alignment.

5. The sequence $x_1 \dots x_k$ with all "_" characters removed is S.

6. The sequence $y_1 \dots y_k$ with all "_" characters removed is S.

Identifying Similarity Between Sequences

In general, an instance of the global alignment problem must consider the four cases of aligning a specific character of one sequence to a character of another sequence:

1. alignment: the character in string S is aligned with the same character in string T;

2. replacement: the character in S is aligned with a **different** character in T;

3. deletion: the character in S is NOT aligned with any character in T (essentially, removed from alignment);

4. insertion: a character from T is NOT aligned with any character in S (essentially, it was inserted).

Given an alignment $a(S,T) = \{(x_1, y_1), \dots (x_k, y_k)\}$, we associate with each pair (x_i, y_i) of aligned characters its score $score(x_i, y_i)$:

- Higher scores: better alignment.

- Lower scores: poorer alignment.

The total score of an alignment is defined as the sum of all pairwise scores:

$$Score(a(S,T)) = \sum_{i=1}^{k} score(x_i, y_i).$$

Example. The **edit distance** problem is an instance of a global alignment problem where the following scoring mechanism is established.

1. matching a character in a position has a score of 0;

2. replacing a character has a score of -1;

3. deleting a character has a score of -1;

4. inserting a character has a score of -1.

Scoring function (scoring matrix, substitution matrix). Let Σ be an alphabet, and "_" $\notin \Sigma$. Let $\Sigma' = \Sigma \cup \{\text{"\_"}\}$. A function

$$score : \Sigma' \times \Sigma' \rightarrow \mathcal{R}$$

which matches each pair of characters in σ' to a numeric score is called a **scoring function** or a **scoring matrix**.

In bioinformatics, two different alphabets are used in sequence alignment problems: the nucleotide alphabet and the amino acid alphabet.

Substitution matrices for nucleotide alphabet. For the nucleotide alphabet, usually, only two numbers are specified: one for the cost of a match, and one for the cost of mismatch/deletion/insertion. For global alignment, the following two variants can be used:

	A	T	C	G	_
A	1	0	0	0	0
T	0	1	0	0	0
C	0	0	1	0	0
G	0	0	0	1	0
_	0	0	0	0	$-\infty$

	A	T	C	G	_
A	5	-4	-4	-4	-4
T	-4	5	-4	-4	-4
C	-4	-4	5	-4	-4
G	-4	-4	-4	5	-4
_	-4	-4	-4	-4	$-\infty$

Substitution matrices for amino acid alphabet. There are two families of matrices, known as PAM (Point Accepted Mutation) and BLOSUM (BLOcks of amino acid SUbstitution Matrix), that use changes in amino acid structure in knows sequence collections and use somewhat different means of estimating the likliehood of a replacement. More probable replacements score higher, less probably replacements score lower. (See a separate handout for more information on the PAM and BLOSUM matrices and their most popular versions).

NeedlemanWunsch algorithm for Global Alignment

Problem. Given two strings $S = s_1 \dots s_n$ and $T = t_1 \dots t_m$ and a substitution matrix $score[i, j]$, return an alignment $a(S, T)$ with the highest total score.

Needleman-Wunsch algorithm. Needlman-Wunsch algorithm, proposed in 1970 is a dynamic programming algorithm that, essentially extends the edit distance algorithm (and the LCS algorithm).

The algorithm works in two steps:

1. Compute table $c[i, j]$ of the total score of matching prefixes S_i and T_j for each pair (i, j).
2. Compute the global alignment from the information in $c[i, j]$.

Note. In most subsitution matrices, the cost of an insertion/deletion does not depend on which character is being inserted/deleted, i.e., $cost[\_, y]$ and $cost[x, \_]$ are the same for all $x, y \in \Sigma$. Let $cost[x, \_] = cost[\_, y]$ be denoted as d.

Recurrence for $c[i, j]$. For $0 \le i \le n$ and $0 \le j \le m$, $c[i, j]$ can be defined as follows:

$$c[i,j] = \begin{cases} d \cdot j & \text{if } i = 0; \\ d \cdot i & \text{if } j = 0; \\ \max(c[i, j-1] + d, c[j, i-1] + d, c[i-1, j-1] + score[s_i, t_j]) & \text{if } i, j > 0. \end{cases}$$

The pseudocode for the Needleman-Wunsch algorithm shown below, fills two matrices: $c[i, j]$ as described above, and $u[i, j]$: the matrix that contains information about the best alignment. $u[i, j] \in \{\nwarrow, \leftarrow, \uparrow\}$.

```
Algorithm NWGlobalAlign(S = s_1 ... s_n, T = t_1 ... t_m, score[], d)
begin
  declare c[0..n, 0..m];
  declare u[0..n, 0..m];
  for i = 0 to n do
    c[i, 0] := d · i;
  end for
  for j = 1 to m do
    c[0, j] := d · j;
  end for
  for i = 1 to n do
    for j = 1 to m do
      Replace := c[i − 1, j − 1] + score[s_i, t_j];
      Insert := c[i, j − 1] + d;
      Delete := c[i − 1, j] + d;
      c[i, j] := max(Replace, Insert, Delete);
     if c[i,j] = Replace then
      u[i,j]:= ↖;
      else if c[i, j] = Insert then
       u[i, j] :=←;
      else if c[i, j] = Delete then
       u[i, j] :=↑;
      end if
    end for
  end for
  AlignmentScore:= c[n, m];
  Alignment:= AlignmentRecover(S, T, u[]);
  return (Alignment);
end
```

The algorithm AlignmentRecover takes as input two strings, S and T and the matrix $u[i, j]$ that encodes how $c[i, j]$ was filled, and returns back the alignment of S and T. The algorithm works as follows.

```
Algorithm AlignmentRecover(S = s_1 ... s_n, T = t_1 ... t_n, u[0..n, 0..m])
begin
  A := ∅;
  i := n;
  j := m;
  while  i + j > 0  do        // keep building alignment until c[0,0] is reached
   if  u[i, j] =↖  then        // replacement
     A := A ∪ {(s_i, t_j)};
     i := i − 1;
     j := j − 1;
   else if u[i, j] =←  then        // insertion
     A := A ∪ {("_", t_j)};
     j := j − 1;
   else          // u[i,j] = ↑; deletion
     A := A ∪ {(s_i, "_")};
     i := i − 1;
   end if
  end while
  return A;
end
```

Analysis. Algorithm NWGlobalAlign is essentially an extension of the LCS and EditDistance algorithms. As such, its algorithmic complexity is $O(mn)$.