Computational Biology

Lecture 4

Saad Mneimneh

Semiglobal Alignment

- We score alignments ignoring start and end gaps.
- Start gaps occur before the first character in a sequence.
- End gaps occur after the last character in a sequence

```
CAGCA-CTTGGATTCTCGG
---CAGCGTGG------
```

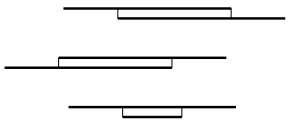
start gaps end gaps

- Score = +1(6) -1(1) -2(1) -11(0) = 3
- This is not the best global alignment between the two sequences (-12 for global optimal against -19 if scored globally)

Saad Mneimneh

Why this variation?

- Maybe it is OK to have unlimited number of gaps at the ends, e.g. detecting significant overlap
- Different possible overlaps

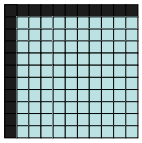


Saad Mneimneh

Modifying Needleman-Wunsch

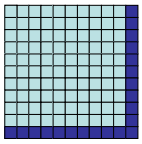
- Ignoring start gaps
  - $A(i,0) = 0$
  - $A(0,j) = 0$

Why?



- Ignoring end gaps

$$A^{OPT} = \max \begin{cases} \max_j A(m,j) \\ \max_i A(i,n) \end{cases}$$



Saad Mneimneh

Explanation

- Initializing the first row and first column to zeros eliminates any starting gap penalty.
- An alignment with end gaps in x aligns x with a prefix of y.

  …xxxxxxx----
  …yyyyyyyyyyy

  Therefore, the $\max_j A(m,j)$ (last row) gives the maximum score for such an alignment.
- Same reasoning for y.

Saad Mneimneh

Generalization

Place where gaps are not penalized	Action
Start of x	Initialize first row to zeros
End of x	Look for max in last row
Start of y	Initialize first column to zeros
End of y	Look for max in last column

Saad Mneimneh

Local Alignment

- Given two sequences *x* and *y*, find a highest score alignment between a substring of *x* and a substring of *y*
- Example:

Global scores -1

```
GGAGTA
GC-GTC
```

Local scores 2

```
G GAGT A
  GCGT C

GGA GT A
 GC GT C
```


Why Local Alignments?

- Comparative genomics:
  - Genes are shuffled in different genomes
- Finding preserved sequences
  - Different proteins have preserved repetitive patterns

Greedy Way

- $O(m^2)$ substrings of *x*
- $O(n^2)$ substrings of *y*
- $O(m^3n^3)$ algorithm using Needleman-Wunsch on each pair

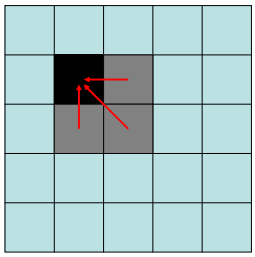
Facts

- The two substrings with the best score must be matched at both ends (why?)
- The score of the best local alignment has to be positive (why?)
- **More generally**: The best local alignment cannot start or end with an alignment with negative score (why?)

Modifying Needleman-Wunsch

- Same as before, but
  - $A(i,j)$ is now best score for aligning a suffix of $x_1 \ldots x_i$ and a suffix of $y_1 \ldots y_j$.
  - Is the update rule still valid? Almost…
  - Negative entries in the A matrix are meaningless
    if $A(i,j) < 0$, replace it with 0
    i.e, starting over from $A(i+1, j+1)$, $A(i, j+1)$, and $A(i+1,j)$.

Illustration



The best local alignment cannot start with a suffix of $x_1 \ldots x_i$ aligned with a suffix of $y_1 \ldots y_j$.

Smith-Waterman

- Initialization
  - $A(0,j) = \mathbf{0}$
  - $A(i,0) = \mathbf{0}$
- Main iteration

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

- Termination

  $A^{OPT} = \max_{i,j} A(i,j)$

Example

		G	C	G	T	C
	0	**0**	**0**	**0**	**0**	**0**
G	**0**	1	**0**	1	**0**	**0**
G	**0**	1	0	1	0	**0**
A	**0**	**0**	0	**0**	0	**0**
G	**0**	1	**0**	1	**0**	**0**
T	**0**	**0**	0	**0**	2	0
A	**0**	**0**	**0**	**0**	0	1

■ negative forced to **0** by the update

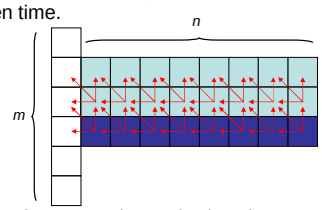
Trace back from $\max_{i,j} A(i,j)$ until you hit a **0**

Saving Space (for Needleman-Wunsch)

- These basic algorithms require $O(mn)$ time and $O(mn)$ space.
- No algorithm is known that uses asymptotically less time and has the same generality.
- It is possible to improve space complexity from $O(mn)$ to $O(m+n)$.

First Attempt

- We only need to keep one row of the matrix A at any given time.



- Therefore, we only need $O(m+n)$ space to compute the optimal score.
- What about producing the alignment itself?

Saving Space (cont.)

- We know how to compute the optimal score in linear space.
  - In fact, we can compute the best scores between a given prefix of x and all prefixes of y (How?)
- The hard part is to obtain the optimal alignment itself in linear space
  - the previous algorithms depend on the whole matrix to do this job.
  - Use a Divide and Conquer to remove this difficulty.

Key Idea

- Consider x_i in an optimal alignment of x and y
  - x_i is aligned with y_j for $1 \leq j \leq n$ OR
  - x_i is aligned to a gap between y_j and y_{j+1} for $0 \leq j \leq n$

. . . . . .
. y_1 . y_2 . y_3 . y_4 . y_5 . … . y_n .

$2n+1$ possibilities

Key Idea (cont.)

Given i, guess what is matched to x_i in an optimal alignment by finding the best alignments that match x_i to each of the $2n+1$ positions

$$\text{OPT}\begin{pmatrix} x_1..x_{i-1} \\ y_1..y_{j-1} \end{pmatrix} \quad \begin{matrix} x_i \\ y_j \end{matrix} \quad \text{OPT}\begin{pmatrix} x_{i+1}..x_m \\ y_{j+1}..y_n \end{pmatrix}$$

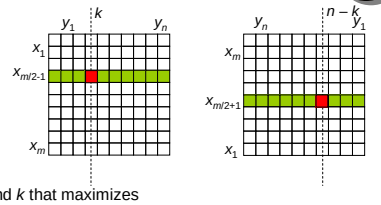
$$\text{OPT}\begin{pmatrix} x_1..x_{i-1} \\ y_1..y_j \end{pmatrix} \quad \begin{matrix} x_i \\ - \end{matrix} \quad \text{OPT}\begin{pmatrix} x_{i+1}..x_m \\ y_{j+1}..y_n \end{pmatrix}$$

Saad Mneimneh

Key Idea (cont.)

- Compute the best scores between
  - $x_1..x_{m/2-1}$ and all prefixes of y
  - $x_{m/2+1}...x_m$ and all suffixes of y (the reverse)

Total time = $c_1.m/2.n + c_1.m/2.n + c_2.n < cmn$



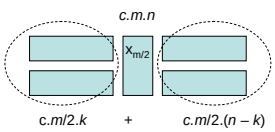
- Find k that maximizes

$$\max \begin{cases} A(m/2-1, k-1) + s(x_{m/2}, y_k) + B(m/2+1, n-k) \\ A(m/2-1, k) + s(x_{m/2}, -) + B(m/2+1, N-k) \end{cases}$$

Saad Mneimneh

Divide and Conquer

- Align $x_{m/2}$
- Recursively do the same for the two remaining chunks of the alignment.



Saad Mneimneh

Analysis

- $T(m,n)$ = time to align $x_1...x_m$ and $y_1..y_n$
- $T(m,n) = c.mn + T(m/2, k) + T(m/2, n - k)$
- Assume $T(m,n) \leq 2c.mn$, verify by substitution:
- $T(m,n) \leq c.mn + 2c.m/2.k + 2c.m/2.(n - k)$
  $= c.mn + 2c.m/2(k + n - k)$
  $= c.mn + 2c.m/2.n = c.mn + c.mn$
  $= 2c.mn$