Lecture Title and Date

Sequence Comparison
02/05/2025

Objectives of the Lecture

- Understand sequence alignment principles and learn about existing alignment techniques
- Understand how to compute and interpret similarity matrices/dot plots
- Understand how to compute sum matrices from similarity matrices and recover the best alignments from a sum matrix (Needleman-Wunsch)

Key Concepts and Definitions

Basic Alignment: Comparing two sequences (can be nucleotides or amino acids) to identify and quantify the similarity between them.

Dynamic Programming: Breaking down bigger problems into smaller subproblems and storing results, so they do not have to be re-computed. This saves computational cost and time.

Gap Penalties: Numerical scores assigned to gaps (indels) to discourage excessive gaps in the alignment process and optimize for biologically realistic/meaningful alignments. Gap penalties can be constant, linear, or otherwise defined.

Global Alignment: Aligns two sequences from end to end, optimizing over the whole length (e.g. Needleman-Wunsch algorithm).

Local Alignment: Identifies subsequences that are the most similar between two sequences, focusing on high-similarity regions rather than the entire sequence (e.g. Smith-Waterman algorithm).

Suboptimal alignment: A sequence alignment that is not the optimal one according to the scoring scheme but still has a high score and may be biologically more relevant than the “optimal” alignment.

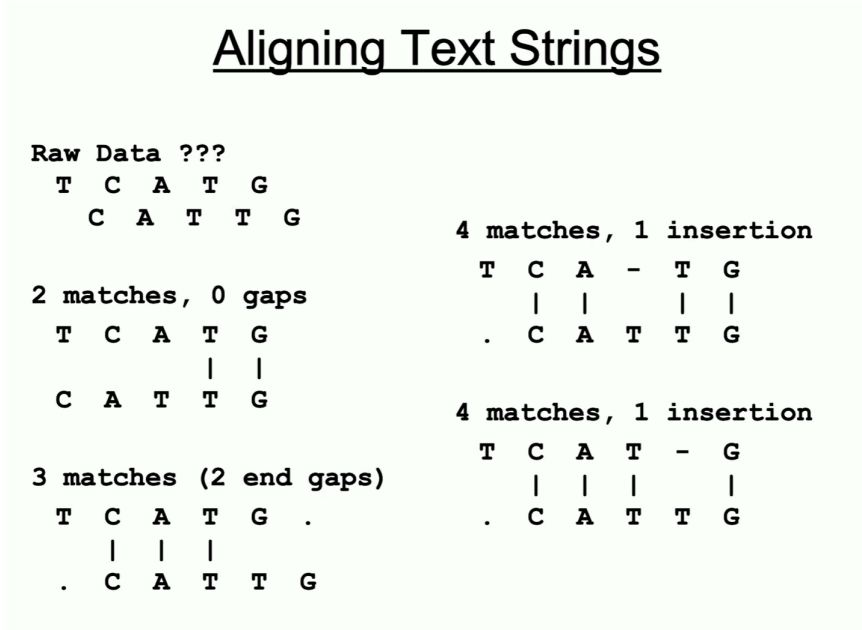
Needleman-Wunsch (1970) automatic alignment method: The first automatic alignment method which uses dynamic programming, similarity matrices/dot plots, and sum matrices to determine the best alignments for two strings.

Similarity matrix/dot plot: A matrix to compare the possible letter-to-letter matches of two strings, with a 1 or a dot indicating that there is a match between two letters and a 0 or an empty cell indicating that two letters do not match.

Sum matrix: a matrix computed from a similarity matrix/dot plot to determine what the highest number of letter-to-letter matches there is for the alignment of two sequences. The best alignments can be recovered from this matrix by tracing back the steps taken to compute the matrix.

Main Content/Topics

In this lecture, Dr. Gerstein wants us to understand some of the basic concepts in sequence alignment. In the image below, he gives an example of how someone may align two small text strings by hand. In the example, if you do not allow for the possibility of gaps and simply align the text strings relative to each other, you achieve two matches. But, if you allow for gaps, you may be able to achieve more matches. For example, if you allow there to be end gaps, then the sequences may only line up in the middle and the end of one sequence could line up with a gap in the other. You could also have insertions, where a letter was inserted into one sequence. During alignment, you would add a gap in the other sequence in the same spot to account for the insertion.



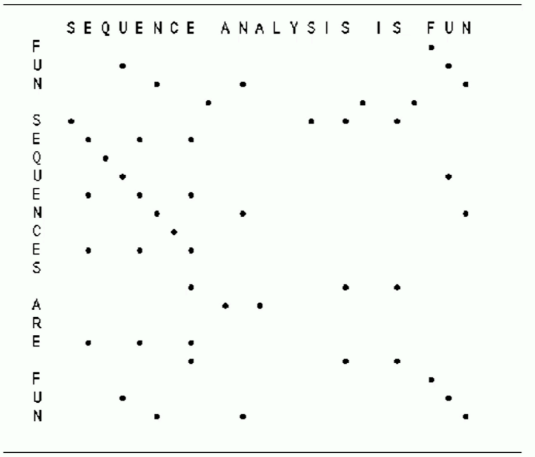
When scientists first started sequencing proteins in the 1960s, they had to deal with much longer strings and could not rely on aligning by hand. Needleman-Wunsch (1970) figured out the first automatic method for alignment, using dynamic programming to find the global alignment of two strings.

Dr. Gerstein then described the steps in the Needleman-Wunsch alignment method:

Step 1: Make a dot plot (similarity matrix)

To make a similarity matrix, you place your two strings horizontally on the top and vertically on the left of a grid. Then you place a **1** in any box if the letters match up in the corresponding row and column, shown in the left figure. The figure on the right shows the **1**s as dots in a dot plot. Importantly, we see that if you have a long string of letters that match, you get a diagonal pattern in your dot plot, as seen with “SEQUENCE” and “FUN”, with “FUN” actually matching in two locations.

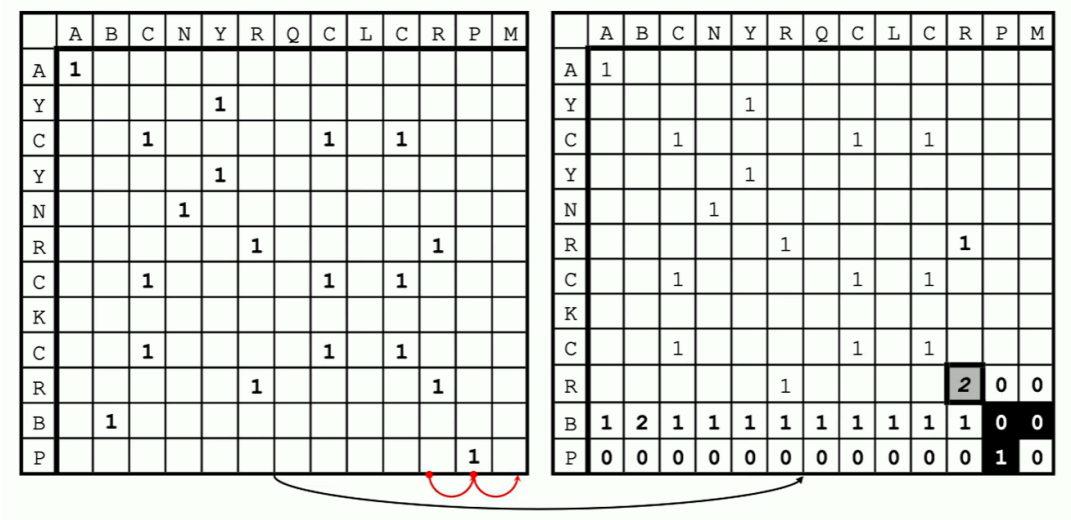
	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	**1**												
Y					**1**								
C			**1**					**1**		**1**			
Y					**1**								
N				**1**									
R						**1**					**1**		
C			**1**					**1**		**1**			
K													
C			**1**					**1**		**1**			
R						**1**					**1**		
B		**1**											
P												**1**	



Step 2: Start computing the sum matrix

We then use the similarity matrix we calculated in **Step 1** (left matrix in the below figure) to compute the sum matrix (right matrix in the below figure). The intuition behind computing the sum matrix is that at each letter-to-letter matching step during alignment, we could either match the letters directly in the two sequences or we could add a gap to either sequence. At each letter-to-letter matching step, we choose the best option (no gap, row gap, or column gap) that results in the best score up to that point, and the option we choose affects how we construct the sum matrix.

The pseudocode below the figures is very helpful for explaining how we construct the sum matrix from the similarity matrix. Essentially, we start with the similarity matrix as our sum matrix, which is a matrix of **1**s and **0**s. Then, we begin in the bottom right corner of the sum matrix and travel left across the bottom row. At each cell, the new value of the cell in the sum matrix is the original value (**1** or **0**) plus the score from choosing the best option (no gap, row gap, column gap). To choose the best option, you take the max of: [the cell diagonally one row down and one column to the right, all cells one row down and at least two columns to the right, and all cells one column to the right and at least two rows down]. The first option corresponds to no gap, the second to making a column gap, and the third to making a row gap.



	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						1					1		
C			1					1		1			
K													
C			1					1		1			
R						1					1		
B		1											
P												1	

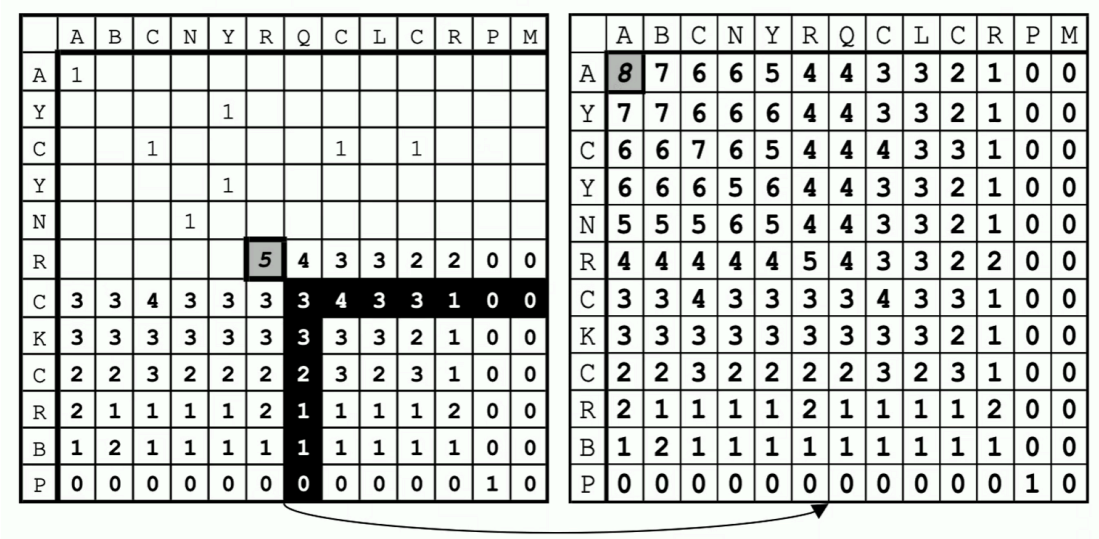
	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						1					1		
C			1					1		1			
K													
C			1					1		1			
R						1					2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

```
new_value_cell(R,C) <=
  cell(R,C)                        { Old value, either 1 or 0    }
  + Max[
       cell (R+1, C+1),            { Diagonally Down, no gaps    }
       cells(R+1, C+2 to C_max),{ Down a row, making col. gap }
       cells(R+2 to R_max, C+1) { Down a col., making row gap }
      ]
```

Step 3: Continue to compute sum matrix and **Step 4: Sum matrix done**

The left matrix in the figure below shows **Step 3**, where we continue to compute the sum matrix. In that matrix, the cell with a **5** is highlighted as the cell we are looking at at that step. The black boxes with white numbers are the cells we look at to determine what the value of the cell in question should be. The original value of the cell in question in the similarity matrix was a **1**, and the max of all the black boxes is a **4**. Therefore, the new value of the cell in question is **1** + **4** = **5**. Since the cell with a **4** was down one row and over two columns, this corresponds to making a column gap.

The right matrix in the figure below shows **Step 4**, which is the completed sum matrix. The highest value (which happens to be in the top left cell, although it doesn't have to be) is an **8**. This means that the most letter-to-letter matches we can get from any alignment is **8**.



	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	1												
Y					1								
C			1					1		1			
Y					1								
N				1									
R						5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

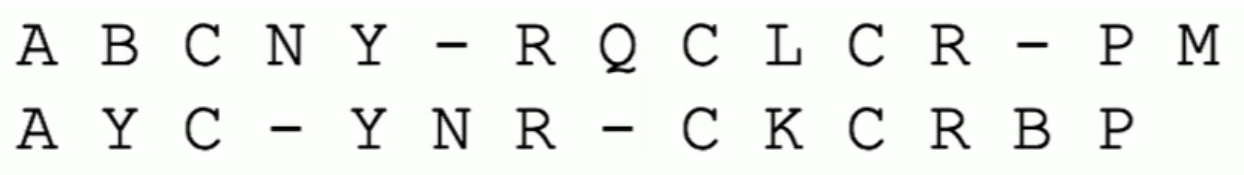
Step 5: Traceback

Now we want to recover the actual alignment that results in the most matches. From our best score, the **8**, we trace back diagonally down and to the right going towards where we started in the bottom right corner. By doing the traceback, we are trying to figure out where we added gaps and to which sequence we added them, and this depends on “remembering” which cell we chose as the “max” in each step when computing the new values of our cells.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

In the example above, we start at the cell with the **8**, since it is the largest one, so we start both of our sequences with **A**. We then look at all of the cells that were considered when figuring out how to calculate the score for the **8** cell (recall from the figure under **Step 3** and **Step 4**, the black cells in the left matrix are the ones we were considering to calculate the **5** cell, so a similar logic can be used to remember which cells we considered for the **8** cell). By looking at these

cells, we see that the **7** is the largest one, so we must have calculated the **8** cell from that one (**7** + **1** = **8**). So, we traceback to the **7** cell that is diagonally touching the **8**. Since the **7** is diagonally touching the **8**, that means that no gap was added, and **AB** in the sequence on the top of the matrix matches directly to the **AY** in the sequence on the left. Then we do the same logic for the **7** we just traced back to, and we trace back to the next **7** cell, so now we have **ABC** matching to **AYC**. When we do the next traceback for that **7** cell, we see that the largest value was the **6** that was two columns over instead of just one (if you notice, there are actually two possible **6**s we could choose, and this conflict will be discussed in **Step 6**). This means that we added a column gap there, so there is a gap between the **Y** and **C** of the sequence on the left of the matrix, and the gap lines up with the **N** of the sequence on the top. So now we have **ABCNY** matching to **AYC–Y**. By continuing this process, we can recover the final alignment show in the image below:



Step 6: Alternate tracebacks

It is also possible that there could be multiple ways to trace back, so there could be multiple alignments that result in the same highest possible number of letter-to-letter matches. In the example we have been using, you can see in the image below that instead of picking the **6** we chose before when tracing back from the last **7**, we could have chosen the **6** that was two rows down and one column over instead. In the forward process, either **6** could have been the “max” that we used to compute the **7**. Therefore, we could form a different alignment by using this **6** instead of the other one. If we use this **6**, we add a gap between **C** and **N** in the sequence at the top of the matrix at the position of the second **y** in the sequence at the left of the matrix. So, we are now matching **ABC–N** to **AYCYN**, which is different from what we did under **Step 5**. Basically, we added a gap in a different spot to get a different alignment, but this alignment will still result in **8** letter-to-letter matches, which is the best we can do. If you notice, we will also add a gap in a different spot when tracing back to the **5** now that the **6**s are in different spots, but after that, we follow the same traceback from **Step 5**. The final alignment from this traceback is shown under the matrix below.

	A	B	C	N	Y	R	Q	C	L	C	R	P	M
A	8	7	6	6	5	4	4	3	3	2	1	0	0
Y	7	7	6	6	6	4	4	3	3	2	1	0	0
C	6	6	7	6	5	4	4	4	3	3	1	0	0
Y	6	6	6	5	6	4	4	3	3	2	1	0	0
N	5	5	5	6	5	4	4	3	3	2	1	0	0
R	4	4	4	4	4	5	4	3	3	2	2	0	0
C	3	3	4	3	3	3	3	4	3	3	1	0	0
K	3	3	3	3	3	3	3	3	3	2	1	0	0
C	2	2	3	2	2	2	2	3	2	3	1	0	0
R	2	1	1	1	1	2	1	1	1	1	2	0	0
B	1	2	1	1	1	1	1	1	1	1	1	0	0
P	0	0	0	0	0	0	0	0	0	0	0	1	0

A B C - N Y R Q C L C R - P M

A Y C Y N - R - C K C R B P

Discussion/Comments

- Needleman-Wunsch algorithm has a time complexity of O(m*n) → high computational cost → important to consider for large-scale data, may not be the best approach
    - Newer tools like FASTA and BLAST may not be as accurate but run a lot faster
- Aligning the entire sequence may not always be desired when there is high similarity on multiple short sequences within a long sequence → local alignment tools may be more appropriate in those cases
- The choice of gap penalty scores affects the alignment results, so choosing an inappropriate gap penalty may lead to suboptimal alignment results
- This lecture looks at the simplest form of sequence alignment from Needleman-Wunsch, where matching amino acids get a score of 1, while mismatches get a score of 0. The paper also discusses more advanced methods, which assign scores based on factors like amino acid properties, genetic code similarities, or surrounding sequences. Gaps in the alignment are can also be discouraged using a penalty, which can depend on the gap's size or direction, and gaps are only allowed if the overall alignment score improves enough to justify them.

Suggested readings:

A somewhat old but neat review on alignment tools and concepts for a general overview:

Vingron, M., & Waterman, M. S. (1994). Sequence alignment and penalty choice: Review of concepts, case studies and implications. *Journal of molecular biology*, *235*(1), 1-12. https://doi.org/10.1016/S0022-2836(05)80006-3

A more recent review on sequence alignment tools, which may be helpful in determining what newer tools and algorithms are available and ready to use:
Chao, J., Tang, F., & Xu, L. (2022). Developments in Algorithms for Sequence Alignment: A Review. *Biomolecules, 12*(4), 546. https://doi.org/10.3390/biom12040546

Other Suggest references for many of the key concepts

Original paper on Needleman-Wunsch algorithm (global alignment):
Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology, 48*(3), 443-453. https://doi.org/10.1016/0022-2836(70)90057-4

Original paper on Smith-Waterman algorithm (local alignment):
Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, *147*(1), 195-197. https://doi.org/10.1016/0022-2836(81)90087-5

Newer tools (prioritize speed → often used for large data):

- FASTA
    - Pearson, W. R., & Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, *85*(8), 2444-2448. https://doi.org/10.1073/pnas.85.8.2444
    - FASTA manual: http://fasta.bioch.virginia.edu/fasta_www2/fasta_list2.shtml
- BLAST
    - Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, *215*(3), 403-410. https://doi.org/10.1016/S0022-2836(05)80360-2
    - BLAST web tool: https://blast.ncbi.nlm.nih.gov/Blast.cgi

Interactive app to visualize alignment matrix for custom sequences using the Needleman-Wunsch algorithm by the Grant Lab at UCSD:
https://bioboot.github.io/bimm143_W20/class-material/nw/