Pre-class assignment

- Submit a PDF (not word doc, please), preferably via Slack
  - [YYYY-MM-DD]_[LastName]_report.pdf
- No need for elaborate headings at the beginning of the report.
  - [Name - Due-Date - Paper title] is enough.
  - The point is to not encroach Page1.
- Page settings:
  - Font: Arial 10pt or Times New Roman 11pt.
  - Margins & spacing: 1-inch margins on all sides; 1 or 1.15 spacing between lines.
- Read the paper & write the report critically but your report is not a peer-review.

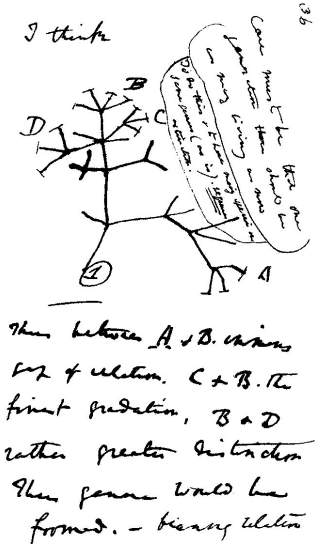
Paper discussion

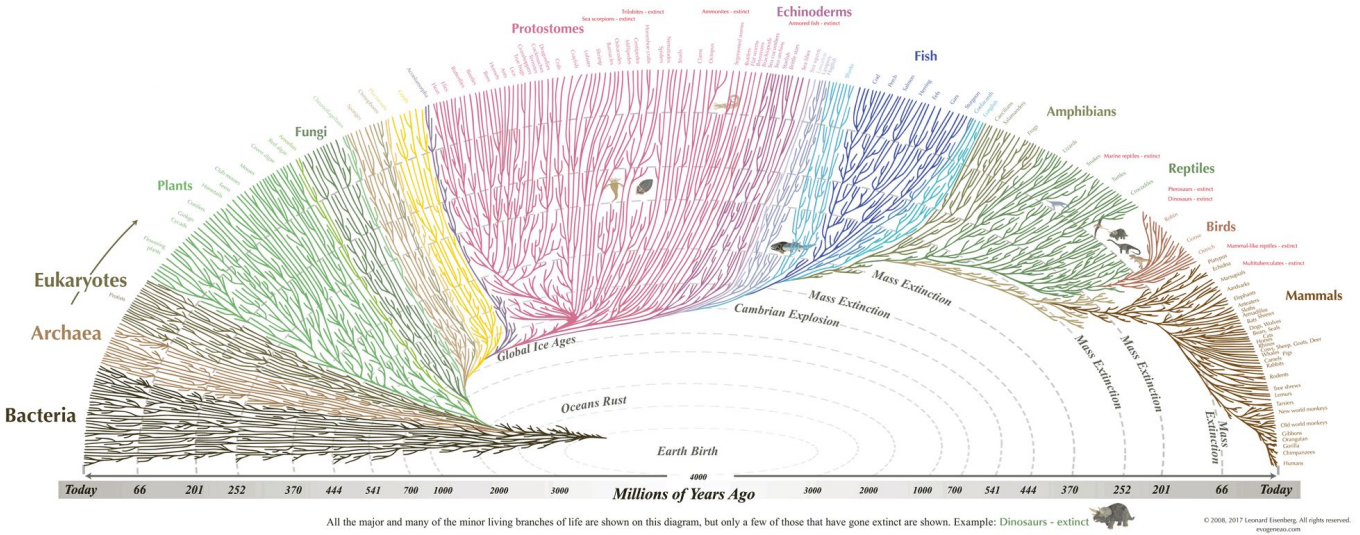
- Use other resources beyond the paper to illustrate the problem.
  - Provide citation on the slide.
- Split complex multi-part figures into several slides.
  - Capture/zoom-into parts of figure & annotate with boxes/arrows.
- Convert methods to flowcharts & annotate flowcharts.
  - You're welcome to draw neatly on paper & photograph it.
- Be prepared to define phrases/terminology on your slide.
  - Google it, read-up; No issues if it's still unclear. Please raise it in class & we can discuss.
- Look-up other papers (esp. recent ones that cite this paper to see what's said).
  - Use that to final discussion about limitations & future directions.

Lecture 4: Sequence alignment

- Global alignment
  - Dynamic programming
  - Needleman-Wunsch algorithm
- Local alignment
  - Smith-Waterman algorithm
  - BLAST

Sequence evolution





deletion mutation insertion

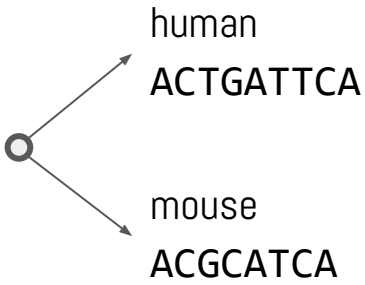
ACATGGTCA → AC*TGGTCA → ACTG**A**TCA → ACTGAT**T**CA

Evolutionary time

human
ACTGATTCA

mouse
ACGCATCA

Sequence evolution



Sequences can be aligned by allowing for **gaps** and **mismatches**.

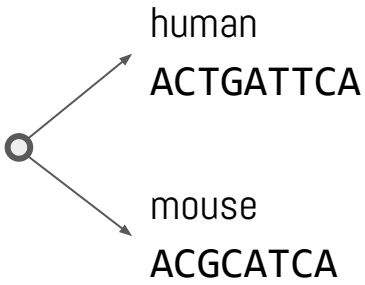
```
ACTGATTCA
ACGCA-TCA
```

```
ACTGATTCA
AC-GCATCA
```

```
ACTG-ATTCA
AC-GCAT-CA
```

Which alignment is correct?

Sequence evolution



Sequences can be aligned by allowing for **gaps** and **mismatches**.

```
ACTGATTCA
ACGCA-TCA
```

```
ACTGATTCA
AC-GCATCA
```

```
ACTG-ATTCA
AC-GCAT-CA
```

Which alignment is correct?

A scoring scheme:

- Match: 2
- Mismatch: -3
- Gap: -2

2+2-3-3+2-2+2+2+2
= 4

2+2-2+2-3-3+2+2+2
= 4

2+2-2+2-2+2+2-2+2+2
= 8

Alignment is gap placement.
How many possible alignment?

Dynamic programming

Solve a given complex problem by breaking it into **subproblems** and store the results of subproblems to avoid computing the same results again.

Two key properties of a problem that suggest that the given problem can be solved using DP.

1. Overlapping Subproblems
    - Given problem can be recursively broken down into subproblems that can be related to each other. This is total no. of subproblems is polynomial.
2. Optimal Substructure
    - The optimal solution can be produced by combining optimal solutions of subproblems.

Richard Bellman

Optimal decision processes, involved time series & planning - thus 'dynamic' & 'programming'.

"It's impossible to use the word dynamic in a pejorative sense"; DP was "something not even a Congressman could object to."

Dynamic programming

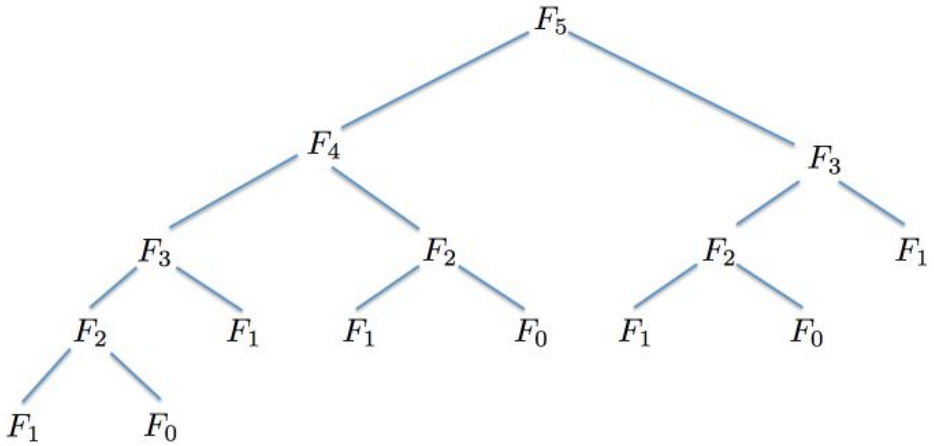
Hemachandra/Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

$$F_0 := 0;\ F_1 := 1;$$

$$F_n = F_{n-1} + F_{n-2},\ \text{for all } n \geq 2.$$

A trivial algorithm for computing F_n:

```
naive_fib(n):
     if n ≤ 1: return n
     else: return naive_fib(n − 1) +
                  naive_fib(n − 2)
```



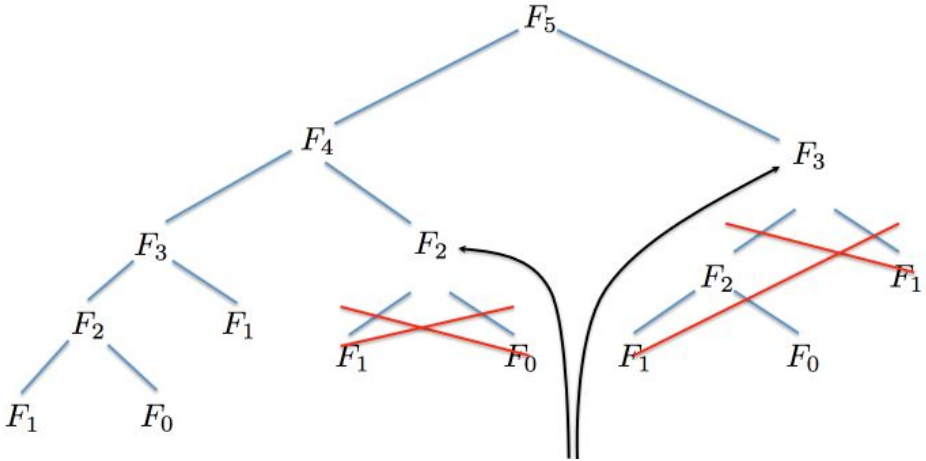
Dynamic programming

Hemachandra/Fibonacci numbers: $F_0 := 0;\ F_1 := 1;\ F_n = F_{n-1} + F_{n-2}$, for all $n \geq 2$.

Never recompute a subproblem $F(k)$, $k \leq n$, if it has been computed before.
Remembering previously computed values: **memoization**.

Improved algorithm for computing F_n:

```
memo = { }
fib(n):
     if n in memo: return memo[n]
     else if n = 0: return 0
     else if n = 1: return 1
     else: f = fib(n − 1) + fib(n − 2)
     memo[n] = f
     return f
```



These values are already computed and stored in memo when runtime processes these nodes of the recursion.

Dynamic programming

1. Overlapping Subproblems
2. Optimal Substructure

DP ≈ recursion + memoization (reuse)

- Remember (memoize) previously solved "subproblems"; e.g., in Fibonacci, we memoized the solutions to the subproblems $F_0, F_1, \cdots, F_{n-1}$, while unraveling the recursion.
- If we encounter a subproblem that has already been solved, reuse solution.
- Runtime ≈ (no. of subproblems) * (time per subproblem)

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **GCAT** with **GAT**

A scoring scheme:

- Match: 1
- Mismatch: -1
- Gap: -1

	–	G	C	A	T
–					
G					
A					
T					

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **GCAT** with **GAT**

$$M(0, j) = j*p$$

$$M(i, 0) = i*p$$

$$M(i, j) = \mathrm{MAX}\begin{cases} M(i-1, j) + p & \text{top} \\ M(i, j-1) + p & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

	–	G	C	A	T
–					
G					
A					
T					

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **GCAT** with **GAT**

$$M(0, j) = j*p$$

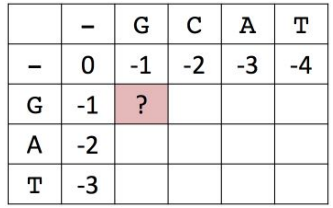
$$M(i, 0) = i*p$$

$$M(i, j) = \mathrm{MAX}\begin{cases} M(i-1, j) + p & \text{top} \\ M(i, j-1) + p & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

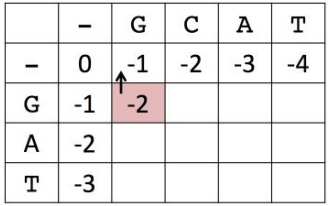
	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1				
A	-2				
T	-3				

Needleman-Wunsch algorithm

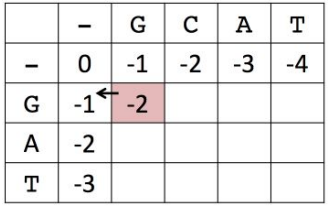
1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback



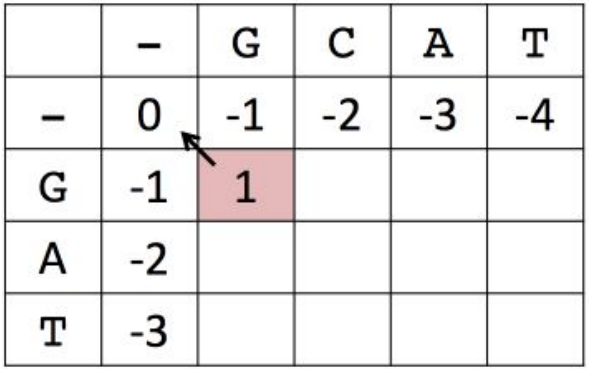
	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1	?			
A	-2				
T	-3				



	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1	-2			
A	-2				
T	-3				



	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1	-2			
A	-2				
T	-3				



	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1	1			
A	-2				
T	-3				

```
M(0, j) = j*p
M(i, 0) = i*p

M(i, j) = MAX(   M(i-1, j) + p,        top
                 M(i, j-1) + p,        left
         M(i-1, j-1) + S(A_i, B_j)  )  diagonal
```

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **GCAT** with **GAT**

$$M(0, j) = j*p$$

$$M(i, 0) = i*p$$

$$M(i, j) = \mathrm{MAX}\begin{cases} M(i-1, j) + p & \text{top} \\ M(i, j-1) + p & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

	–	G	C	A	T
–	0	-1	-2	-3	-4
G	-1	1	0	-1	-2
A	-2	0	0	1	0
T	-3	-1	-1	0	2

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **GCAT** with **GAT**

```
GCAT
G-AT
```

$$M(0, j) = j*p$$

$$M(i, 0) = i*p$$

$$M(i, j) = \mathrm{MAX}\begin{cases} M(i-1, j) + p, & \text{top} \\ M(i, j-1) + p, & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

	-	G	C	A	T
-	0	-1	-2	-3	-4
G	-1	1	0	-1	-2
A	-2	0	0	1	0
T	-3	-1	-1	0	2

Needleman-Wunsch algorithm

1. Scoring function: substitution matrix & gap penalty
2. Matrix initialization & filling
3. Traceback

Align **ATGCT** with **ATTACA**

$$M(0, j) = j*p$$

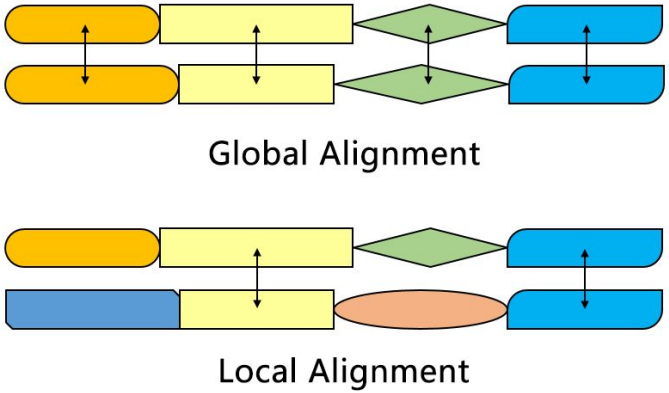
$$M(i, 0) = i*p$$

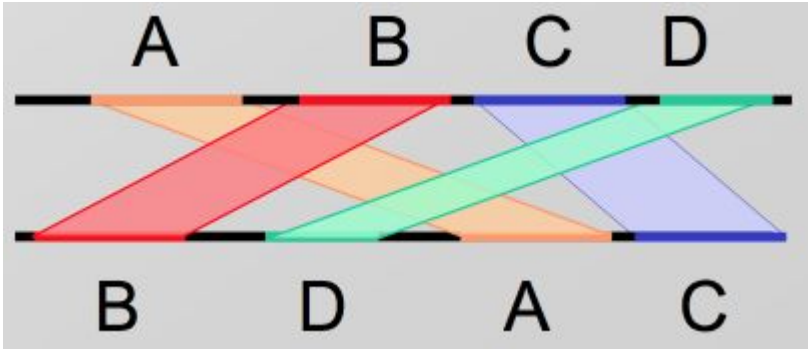
$$M(i, j) = \mathrm{MAX}\begin{cases} M(i-1, j) + p, & \text{top} \\ M(i, j-1) + p, & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

	-	A	T	T	A	C	A
-							
A							
T							
G							
C							
T							

Local alignment

A local alignment of strings *s* and *t* is an alignment of a substring of *s* with a substring of *t*.





Smith-Waterman algorithm

Similar to Needleman-Wunsch, with 3 changes:

- First row/column set to 0.
- No negative scores, set to 0.
- Backtrack from cell with highest score, stop at 0.

Align **GCAT** with **GCT**

$$M(0, j) = 0$$

$$M(i, 0) = 0$$

$$M(i, j) = \mathrm{MAX}\begin{cases} 0 \\ M(i-1, j) + p & \text{top} \\ M(i, j-1) + p & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

Smith-Waterman algorithm

Similar to Needleman-Wunsch, with 3 changes:

- First row/column set to 0.
- No negative scores, set to 0.
- Backtrack from cell with highest score, stop at 0.

$$M(0, j) = 0$$

$$M(i, 0) = 0$$

$$M(i, j) = \mathrm{MAX}\begin{cases} 0 \\ M(i-1, j) + p & \text{top} \\ M(i, j-1) + p & \text{left} \\ M(i-1, j-1) + S(A_i, B_j) & \text{diagonal} \end{cases}$$

Align **GCAT** with **GAT**

GC
GC

	–	G	C	A	T
–	0	0	0	0	0
G	0	1	0	0	0
C	0	0	2	1	0
T	0	0	1	1	2

How do we scale this up to search an entire sequence database?

Given a query sequence, and a large set of target sequences (millions),
which target sequences (if any) are related to the query?

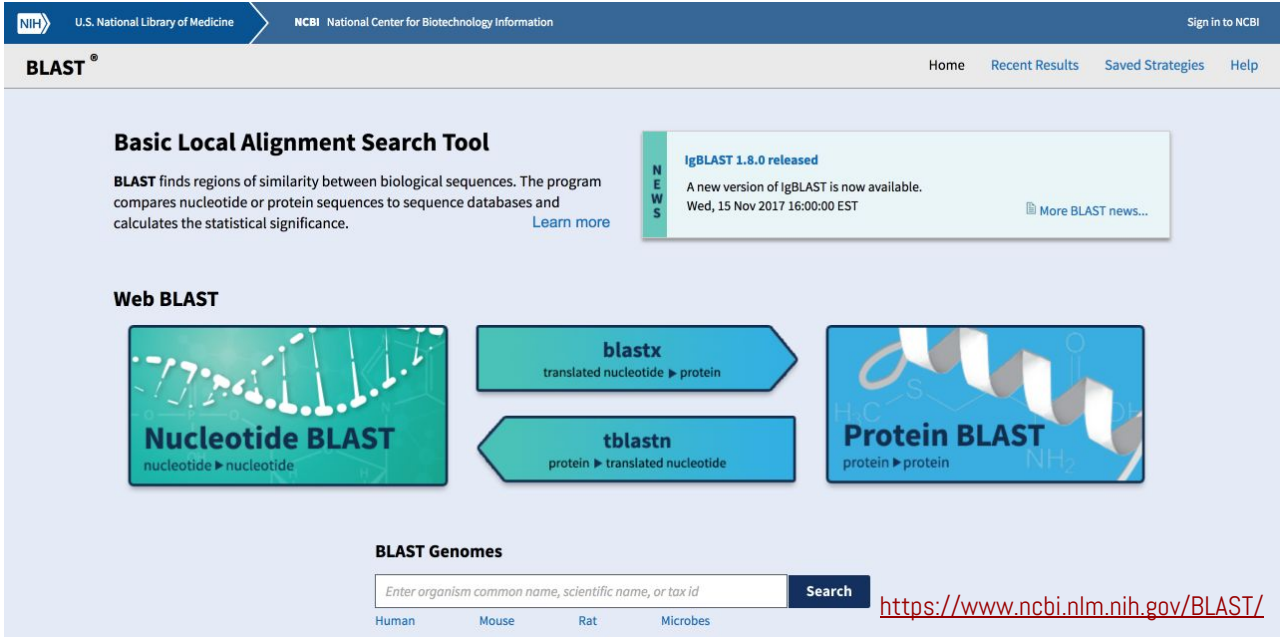
- Individual alignments need not be perfect: Once initial matches are found, they can fine-tune them later.
- Must be very fast.

Exploit the nature of the problem (most sequences will be unrelated to the query):

- If any match with % identity ≤ 90 is going to be rejected, can ignore sequences which don't have a stretch of 10 nucleotides in a row.
- Pre-screen sequences for common long stretches.
- Pre-process the database offline and index k-mers.

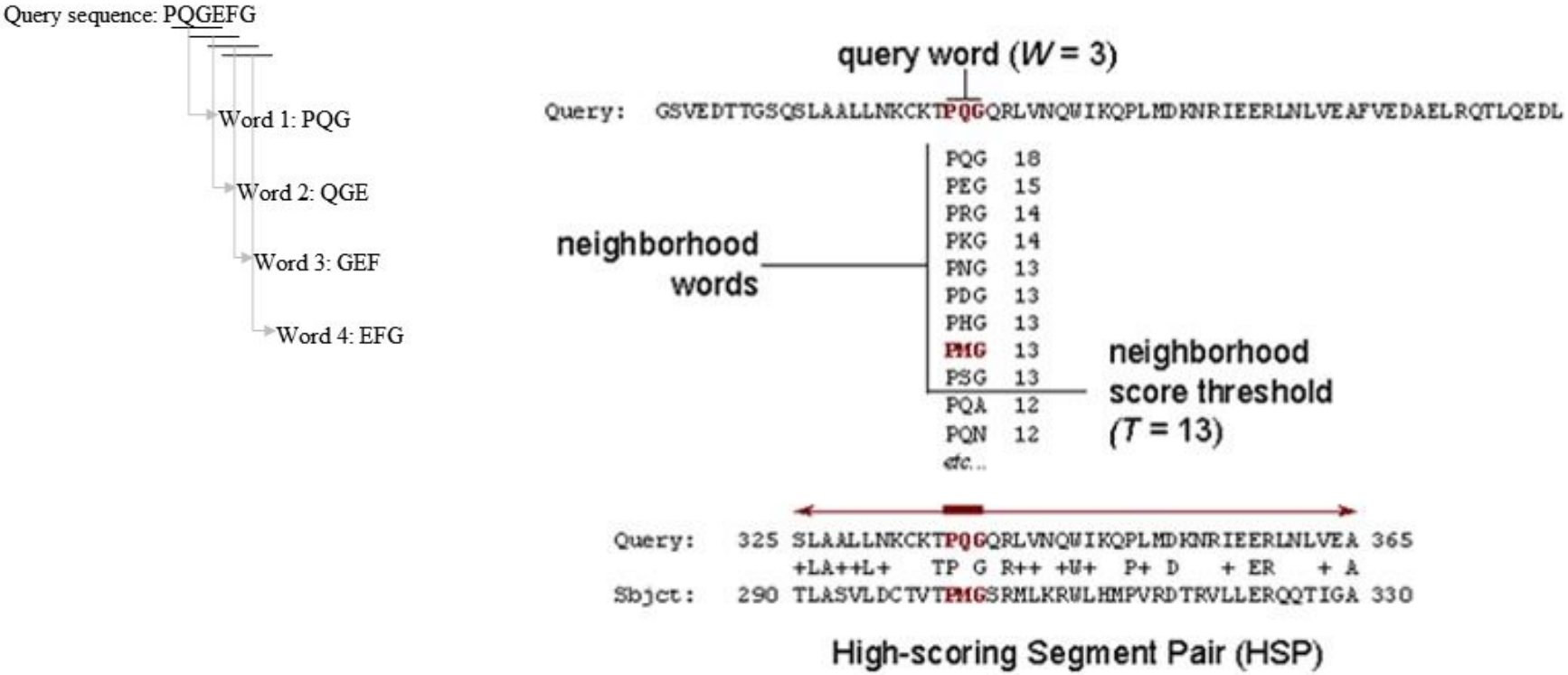
BLAST

TITLE	CITED BY	YEAR
Basic local alignment search tool SF Altschul, W Gish, W Miller, EW Myers, DJ Lipman Journal of molecular biology 215 (3), 403-410	69248	1990
Gapped BLAST and PSI-BLAST: a new generation of protein database search programs SF Altschul, TL Madden, AA Schäffer, J Zhang, Z Zhang, W Miller, ... Nucleic acids research 25 (17), 3389-3402	64876	1997



https://www.ncbi.nlm.nih.gov/BLAST/

BLAST



NCBI Handbook: https://www.ncbi.nlm.nih.gov/books/NBK153387/