Algorithms in Bioinformatics

CS 482

Bin Ma

LaTeXed by Sibelius Peng

Preface

Disclaimer Much of the information on this set of notes is transcribed directly/indirectly from the lectures of CS 482 during Winter 2022 as well as other related resources. I do not make any warranties about the completeness, reliability and accuracy of this set of notes. Use at your own risk.

For any questions, send me an email via https://notes.sibeliusp.com/contact.

You can find my notes for other courses on https://notes.sibeliusp.com/.

Sibelius Peng

Contents

1

Introduction

This course is officially titled as "Computational Techniques in Biological Sequence Analysis". However, this is an old title and this course has been offered for over approximately twenty years. It should actually be called "Algorithms in Bioinformatics". Around 1980, this area has a different name: Computational Biology. You may also hear another name: DNA computing. It is another area, and it does not solve biological problem. What is bioinformatics? It is biology + informatics. Biology is the reason, goal, purpose and informatics is the method.

Biology can be studied at different scales. In old days, biology tries to study organisms, namely living things, such as bacteria, animals. This is because people didn't have tools to study at a lower level at that time. Now people study organs and tissues. Then people can look into cell level, molecular level. At molecular level, DNA is chain of nucleotide bases. Protein is chain of amino acids.

There are a lot of public and free molecular data. There are tremendous amount of public biomolecule data and free software. For example:

- NCBI's sequence data bank: https://www.ncbi.nlm.nih.gov/nuccore/NC_045512
- PDB protein structure database: https://www.rcsb.org/structure/6vxx

Why do people do bioinformatics? The goal is to understand life at molecular level. Especially, for human health. For example, sequencing SARS-Cov2 genomes allowed people study the evolution of this virus. Study the structure of the spike protein and its interaction with the host cells. A lot of human diseases are related to genetics.

There are two areas of bioinformatics.

- Determine the molecule information, by analyzing the data produced by measuring instruments. Typically the data is in large scale and high throughput, which is hard for people to look at.
- Use the molecular data to make inference. For example, make prediction given the existing data.

Consider an example of genome sequencing. From wikipedia,

> The Human Genome Project (HGP) was an international scientific research project with the goal of determining the base pairs that make up human DNA, and of identifying, mapping and sequencing all of the genes of the human genome from both a physical and a functional standpoint. It remains the world's largest collaborative biological project. Planning started after the idea was picked up in 1984 by the US government, the project formally launched in 1990, and was declared complete on April 14, 2003. Level "complete genome" was

achieved in May 2021.

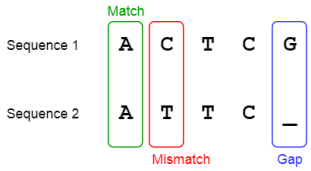
Bioinformatics played an essential role in analyzing the data and assemble the genome. Today one can sequence a human's genome with $<$ \$1000 in a couple of weeks. Bioinformatics is the key to utilize the NGS (next generation sequencing) data for genome sequencing. As such, today's cancer treatment starts to become personalized. And many new drugs now require gene sequencing as companion diagnostic.

1.1 Objectives of this course

- Know bioinformatics
  - Purpose and method
  - General topics
- Learn classic problems and algorithms in bioinformatics
- Learn wide-applicable computational techniques
  - String algorithms
  - Hidden Markov Model
  - Log likelihood ratio score
  - Statistical validation
  - A bit of machine learning

Some typical problems:

- Gene prediction problem
- Find the longest shared substring between human and mouse genomes. If we want to find similarities instead of exact matches, this will lead us to the homology search problem.
- Peptide Identification



Part I:

Sequence Alignment

Img src: https://towardsdatascience.com/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f

2

Sequence Alignment

2.1 Biology

Consider two protein sequences:

- AVP78042.1 spike protein: `MLFFL...`
- YP_009724390.1 surface glycoprotein: `MFVFL...`

They look similar, how do we know these two proteins are similar? It's better to visualize them properly so that we can see the similarity. This is done by sequence alignment, and there are many existing tools: such as Clustal Omega. It's a common belief that these two proteins are developed from a common ancestor, then they evolve from an evolution tree. This is called homology.

There are two classes of nucleotide bases:

- Purine: A and G
- Pyrimidine: T and C

Base pairs are bonded by hydrogen bonds. Also, G-C bind stronger because of 3 H-bonds. DNA molecule is oriented.

So we know that DNA is double-helical, with two complementary strands. And the complementary bases: A-T, G-C. For example, the *reverse* complement of `AAGGTAGC` is `GCTACCTT`, because DNA is oriented.

Now consider DNA mutation. DNA mutates with a small probability when inherited by the offspring. For example, one base can be substituted by another, because there might be copy errors. This creates different alleles of the same gene. From wiki, allele is one of two, or more, forms of a given gene variant. When we inherit the DNA from parents, we only inherit half of each parent's genome. These together cause the differences between individuals of the same species.

Single Nucleotide Polymorphisms is a germline substitution of a single nucleotide at a specific position in the genome. Single base variation between members of a species. For Human, 90% of all human genetic variation is caused by SNPs. SNPs occur every 100 to 300 bases along the 3-billion-base human genome. It's a major risk for genetic disease, because when one base pair mutates, it will cause the express protein's functions.

2.2 Compare DNA sequences

The most often used distance on strings in computer science is Hamming distance. This makes some sense on comparing DNA sequences in some cases: substitution. But there are other mutations: insertion/deletion (indel), which cannot be modelled correctly by Hamming distance. Other DNA rearrangements can also happen. But substitutions and indel are the two mutations we concern the most for this course.

Edit distance

Instead, we can use **edit distance**. How "far" away are two sequences from each other? Edit distance is defined to be the minimum number of edit operations needed to convert one to another. Here edit operations include substitutions and indels.

Note that Edit distance is a distance metric:

- Identity: $d(x, y) = 0$ if and only if $x = y$.
- Symmetry: $d(x, y) = d(y, x)$.
- Triangular inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

Now we prepare for the algorithm. For convenience of the proof, we treat each occurrence of the same letter different. For example, `ATAA -> ATA`, `A` can be done by either deleting the 2nd or 3rd letter `A` from the first string. These are different editing paths. This does not affect our definition of edit distance, but makes our later proof more precise.

Now it's ready to develope the dynamic programming algorithm for edit distance. Let $D[i, j] =$ edit distance between $S[1..i]$ to $T[1..j]$. Consider the edit operations associated with $S[i]$ and $T[j]$ the optimal edit operations. One of the following cases will happen:

1. $S[i]$ is deleted: $D[i, j] = D[i - 1, j] + 1$
2. $T[j]$ is inserted: $D[i, j] = D[i, j - 1] + 1$
3. $S[i]$ becomes $T[j]$: $D[i, j] = D[i - 1, j - 1] + \delta(S[i], T[j])$

where $\delta(S[i], T[j]) = 0$ if $S[i] = T[j]$ and 1 if not.

Algorithm 1: Dynamic Programming Algorithm for Edit distance

```
D[0,0] = 0
D[0,i] = i for i = 1..|S|
D[i,0] = i for i = 1..|T|
for i ← 1..|S| do
    for j ← 1..|T| do
        D[i,j] = min{D[i-1,j] + 1, D[i,j-1] + 1, D[i-1,j-1] + δ(S[i],T[j])}
return D[|S|,|T|]
```

Longest Common Subsequence

Another way to evaluate the similarity of two sequences is through LCS. A subsequence is obtained by deleting some of the letters from the supersequence and concatenating the remaining letters together. For example, LCS of `ATGCATTTA` and `ATGTACTTTC` is `ATGATTT`. LCS can be computed with dynamic programming as well.

2.3 Alignment

The third way to compare to sequences is through sequence alignment. We want to insert spaces (-) to two sequences so that we can align them together and they are most similar column-wisely. For example,

```
ATGCA-TTTA
|||  |  || |
ATGTACTT-A
```

By "similar", we usually need to use a scoring function. We define the alignment score to be **the total of column scores**. And each column is assigned by a constant score depending on matching conditions. For example, simple score scheme would be

- Match = 1
- Mismatch = -1
- indel = -1

Consider two alignments with the score scheme above:

```
AATGCGA-TTTT
||  |  |||
G-TG--ACTTTC
```

has score 0.

```
AATG-CGATTTT
||  |  ||
G-TGAC-TTTC-
```

has score -1.

As a side note, alignment can "simulate" LCS and edit distance. We have this following table for the scoring system:

	alignment	LCS	edit dist
match	+1	1	0
mismatch	-1	0	-1
indel	-1	0	-1

Now we can develop the develop the dynamic algorithm for the alignment. Let $f(a, b)$ be the scoring scheme for a column with a and b. Here one of a and b can be the dash character -. Thus $f(\text{-}, x)$ and $f(x, \text{-})$ represent scores of indels. The input is S, T. Let $D[i, j]$ be the optimal alignment score for $S[1..i], T[1..j]$.

Now we want to derive a recurrence relation. Suppose we are to align $S[1..i], T[1..j]$. Consider the *last column* of the optimal alignment. Three cases can happen.

$S[1..i-1]$ $S[i]$	$S[1..i-1]$ $S[i]$	$S[1..i-1]$ -
$T[1..j-1]$ $T[j]$	$T[1..j-1]$ -	$T[1..j-1]$ $T[j]$
(1)	(2)	(3)

Note that in each case, the sub-alignment without the last column is an optimal one. Prove by contradiction, assume there's a better sub-alignment without the last column. Then the optimal alignment (for the whole) should have used this better sub-alignment, instead of the current sub-alignment.

Then we have the recurrence for these three cases:

(1) $D[i,j] = D[i-1,j-1] + f(S[i],T[j])$

(2) $D[i,j] = D[i-1,j] + f(S[i],\text{-})$

(3) $D[i,j] = D[i,j-1] + f(\text{-},T[j])$

Algorithm 2: Sequence alignment

```
1 D[0,0] = 0
2 for i ← 1..m do
3 |   D[i,0] = i× indel
4 for j ← 1..n do
5 |   D[0,j] = j× indel
6 for i ← 1..m do
7 |   for j ← 1..n do
8 |   |   D[i,j] = max { D[i-1,j-1] + f(S[i],T[j])
  |   |                { D[i-1,j] + f(S[i],-)
  |   |                { D[i,j-1] + f(-,T[j])
9 return D[m,n]
```

Then the time complexity is $O(mn)$ where $|S| = m, |T| = n$. In practice, when we run the DP algorithm and build the DP table, we can add arrows from cell to cell: which of the three cases (sub-alignments) is chosen to get the current cell. Then we can backtrack the arrows and get the actual alignment. For backtracking, it is $O(n+m)$. Space complexity is $O(nm)$.

In practice, we don't need to record/store these arrows. Instead, when we backtrack, we can calculate the arrows based on the max of three options dynamically, which takes $O(m+n)$. This saves the cost of storing arrows, which costs $O(mn)$.

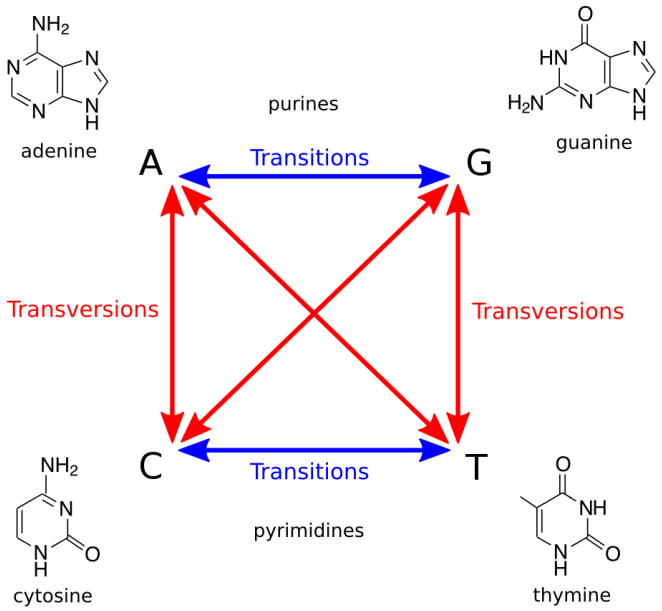
Moreover, if only score is needed, then space complexity can be reduced to linear space.

Note that the algorithm is designed for any score scheme $f(x,y)$. We indeed separate the algorithm and scoring. So we can optimize score scheme and the algorithm independently. Dijkstra once said:

> The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer.

2.4 Score Function

> Transition vs. Transversion
>
> Recall that within DNA, there are two classes. One is purines (A, G), and the other is pyrimidines (C, T). Within the same class, they have similar chemical structure. Moreover, the mutation within the same class is called transitions; across the two classes, is called transversions. Transition is cheaper than transversions.



Picture from https://dodona.ugent.be/en/activities/1351011626/

Transition happens more frequently 2/3 of SNPs are transitions. In other words, transition is easier and therefore should be less penalized. For example,

```
AAAGCAAA        AAAGCAAA
AAAT-AAA   vs   AAA-TAAA
```

Observe that GT is transversion and CT is transition. The right alignment is probably better. This can be easily achieved by changing score scheme $f(a, b)$.

So how to build a score function? First, we need to know what you want. There are two purposes:

1. the optimal alignment reveals the true evolutionary history.
2. high score indicates homology (derived from same ancestor).

We want purpose 1 if possible, but purpose 2 is also useful.

For purpose 1, note that we might be wrong: score function might not have the power to reveal the history. For example, if the history goes as `A -> T -> A`, then in the sequence alignment, the final comparison is between `A` and `A`, which will not give us the true history. So you should never trust that an alignment must be right. It just optimizes the score. Should we give up purpose 1 at all?

For purpose 1, the optimal alignment may be *approximately* correct *under certain conditions* in practice. As long as we know the limitation, we can still use it. For example, for the following alignment, it is "very likely" the alignment is approximately equal to the evolutionary history.

```
ACGTATTACCGG-TTACCG
|||  ||||||||  ||||||
ACGGATTACCGGATTACCG
```

So we should keep in mind that when the score is low, alignment itself is not too useful.

Now let's try to improve our score function. Consider two alignment (with gaps):

```
AGATTTTTTTC     AGATTTTTTTTTC
AGA---TTTTC     AGA-T-T-T-T-C
```

The left seems "simpler" than the right, intuitively. Indeed, during evolution, indels are relatively rare. However, insertion or deletion a segment of k consecutive bases is much easier than k scattered indels. But our current scoring method (adding up column scores) cannot distinguish the two. Currently, a gap of length k costs $k\times$indel. Thus, this is called the **linear gap penalty**. Denote the penalty function by $g(x)$, and it should have negative value. Left's penalty is $g(3)$ and right is $3 \times g(1)$.

Consecutive insertions or deletions are called a gap. Suppose the gap penalty of a length k gap is $g(k)$ instead of the simple $c \times k$. Assume $g(x) + g(y) \leq g(x + y)$, thus a convex function. Otherwise does not serve the purpose of grouping indels. In this case, can the old DP algorithm still work? Recall the three cases:

$S[1..i-1]$ $S[i]$	$S[1..i-1]$ $S[i]$	$S[1..i-1]$ -
$T[1..j-1]$ $T[j]$	$T[1..j-1]$ -	$T[1..j-1]$ $T[j]$
(1)	(2)	(3)

First case is the same. The second case might be wrong. The last column might depend on previous columns. If the second to the last column of T is already a dash, then adding a new dash will increase the length of gap. The we cannot use DP algorithm anymore, in the sense that we cannot prove its correctness. We do not know the contribution of the last column to the gap penalty in the last two cases.

Assume we know the length of gap, then we can apply the recurrence relation. We still use $D[i, j]$ to denote the optimal alignment score of $S[1..i]$ and $T[1..j]$. We change cases 2 and 3 to include the last gap (not the last column). Then three cases change accordingly:

$S[1..i-1]$ $S[i]$	$S[1..i-k]$ $S[i-k+1..i]$	$S[1..i]$ - - - - -
$T[1..j-1]$ $T[j]$	$T[1..j]$ - - - - -	$T[1..j-k]$ $T[j-k+1..j]$
(1)	(2)	(3)

Then $D[i, j] =$ max of the following three cases:

(1) $D[i-1, j-1] + f(S[i], T[j])$

(2) $\max_{1\leq k\leq i} D[i-k, j] + g(k)$

(3) $\max_{1\leq k\leq j} D[i, j-k] + g(k)$

Now the time complexity will change: $O(mn(m+n))$, which is cubic.

In bioinformatics, very often we face the choice between:

- Reality: How close it approximates the real biology.
- Simplicity: How easy it can be computed.

We can simplify $g(k)$ a little bit. We basically want a function that grows slower than linear. We introduce **affine gap penalty**, in contrast to linear gap penalty:

$$g(k) = a + b \cdot k$$

where a is the gap open penalty: whenever we have a gap, we pay for it; b is the gap extension penalty: for example, once we open the gap, we pay less for the second indel within a group.

Example: Affine gap penalty

match $= 1$; mismatch $= -1$; gap open $= -5$; gap extension $= -1$.

```
ATAGG--AAG
||  ||   | |
ATTGGCAATG
```

6 match, 2 mismatch, 1 gap open, 2 gap extension, then the score is

$$6 - 2 + (-5 - 1 \times 2) = -3$$

```
ATAGG-AA-G
||  ||  || |
ATTGGCAATG
```

Similarly, the score is

$$7 - 1 - 5 - 1 - 5 - 1 = -6$$

Now the old algorithm doesn't work anymore. Consider the last column of an alignment again:

```
AT-GG-    ATGG--
||  ||    ||  |
ATTGGC    ATTGGC
```

When considering the last column, we need to know the second to the last column. When the last column is an indel, the added cost depends on the previous column. Because by induction we know the optimal solution for $D[i, j-1]$, we can encode the previous column's configuration. We compute the optimal solution by limiting the last column to one of the following three configurations:

```
ATAGG      ATAGG-     ATAGGC
||  ||     ||  ||     ||  ||
ATTGG      ATTGGC     ATTGG-
```

$D_0[i,j]$ $D_1[i,j]$ $D_2[i,j]$

$D_0[i,j]$ requires that the last column must be $S[i]$ v.s. $T[j]$, not indel. $D_1[i,j]$ is - v.s. $T[j]$, and $D_2[i,j]$ is $S[i]$ v.s. -. We then can develop recursion relationship easier by defining more subproblems. We only distinguish them by the last column, there is no constraint for columns before the last column.

Now let's examine how to calculate $D_0[i,j]$. There are three cases:

$$\begin{matrix} S[1..i-2]S[i-1] & S[i] \\ T[1..j-2]T[j-1] & T[j] \end{matrix} \longrightarrow D_0[i,j] = D_0[i-1,j-1] + f(S[i], T[j])$$

$$\begin{matrix} S[1..i-1] & \text{-} & S[i] \\ T[1..j-2] & T[j-1] & T[j] \end{matrix} \longrightarrow D_0[i,j] = D_1[i-1,j-1] + f(S[i], T[j])$$

$$\begin{matrix} S[1..i-2] & S[i-1] & S[i] \\ T[1..j-1] & \text{-} & T[j] \end{matrix} \longrightarrow D_0[i,j] = D_2[i-1,j-1] + f(S[i], T[j])$$

We then can do the similar thing for $D_1[i,j]$:

$$\begin{matrix} S[1..i-1]S[i] & \text{-} \\ T[1..j-2]T[j-1] & T[j] \end{matrix} \longrightarrow D_1[i,j] = D_0[i,j-1] + \text{gapopen} + \text{gapext}$$

$$\begin{matrix} S[1..i] & \text{-} & \text{-} \\ T[1..j-2] & T[j-1] & T[j] \end{matrix} \longrightarrow D_1[i,j] = D_1[i,j-1] + \text{gapext}$$

$$\begin{matrix} S[1..i-1] & S[i] & \text{-} \\ T[1..j-1] & \text{-} & T[j] \end{matrix} \longrightarrow D_1[i,j] = D_2[i,j-1] + \text{gapopen} + \text{gapext}$$

The relation for $D_2[i,j]$ is symmetric. In summary, we have the recurrence relation

$$D_0[i,j] = f(S[i],T[j]) + \max \begin{cases} D_0[i-1,j-1]; \\ D_1[i-1,j-1]; \\ D_2[i-1,j-1]; \end{cases}$$

$$D_1[i,j] = \text{gapext} + \max \begin{cases} D_0[i,j-1] + \text{gapopen}; \\ D_1[i,j-1]; \\ D_2[i,j-1] + \text{gapopen}; \end{cases}$$

$$D_2[i,j] = \text{gapext} + \max \begin{cases} D_0[i-1,j] + \text{gapopen}; \\ D_1[i-1,j] + \text{gapopen}; \\ D_2[i-1,j]; \end{cases}$$

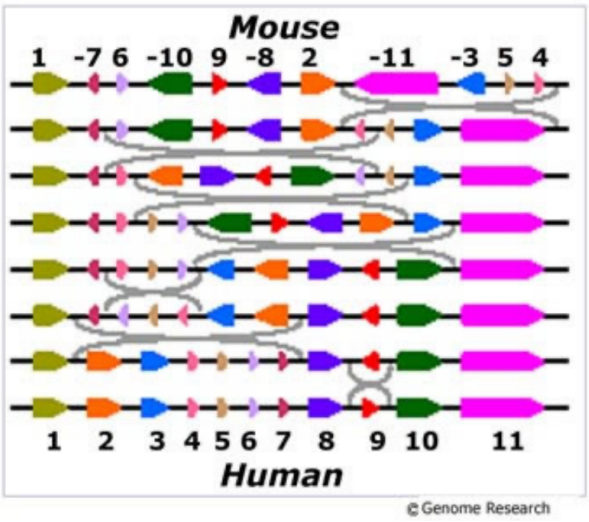
Note the grayed cases can't be optimal so can be safely removed.

This is still DP algorithm. We need to be careful when backtracking. The running time is $O(nm)$, might be approximately 3 times slower, but better than the general gap penalty's cubic time. This is okay because the model is more expressive. This model is first published in 1982, by Gotoh, O.: *An improved algorithm for matching biological sequences*.

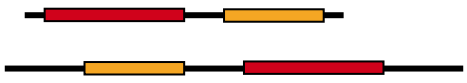
3

Local Alignment and Linear Space Alignment

In biology, we are not comparing S, T directly. Instead, we are comparing small areas of S, T.



Interestingly, if we compare chromosome X of mouse and human, they are quite similar. Each colored block is relatively conserved, but different in orders and orientations. Seven inversions are required to put them in the correct order and orientation. This is called "sorting by reversals". This is an interesting study from Pavel Pevzner.



In this case, if we do the global/sequence alignment, we can align only one part: we will lose yellow part when we align red. Conserved regions are "local" to the genome/chromosome. But previous alignment is "global". We need a proper model to define "local" similarity.

For the problem of local alignment, we are given two sequences S and T. We want to find substrings of S and T that maximizes the alignment score. I.e., The indels at the beginning and end of the two strings are free.

Local alignment score is at least 0, because for the worst case, we have two empty strings. The model only makes sense for alignment but not edit distance nor LCS. Is the optimal local alignment a local part of an optimal "global" alignment? No. If we align `AT` and `TA`, the global alignment would be $\begin{matrix}\texttt{AT}\\ \texttt{TA}\end{matrix}$

if the indel gets the most penalty. While the local alignment would simply be $\begin{matrix}\texttt{A}\\ \texttt{A}\end{matrix}$

3.1 Prefix and suffix alignment

Consider a related different problem: **prefix alignment**: find the highest-scoring alignment between two prefixes of the two sequences. We want to find i, j to maximize alignment-score$(S[i..i], T[1..j])$.

Similarly, consider **suffix alignment**. That is, we choose two suffixes, and align them together optimally. We want to compute $\max_{i',j'}$ score$(S[i'..m], T[j'..n])$

Let $D[i, j]$ denote the optimal "suffix alignment" alignment score of $S[1..i]$, $T[1..j]$. That is, $D[i, j]$ is the maximum alignment score for $S[i'..i]$ and $T[j'..j]$ for all i' and j'. Consider the last column of this optimal "suffix" alignment. Four cases arise:

1. $S[i]$ v.s. $T[j]$
2. $S[i]$ v.s. -
3. $T[j]$ v.s. -
4. an empty alignment

Case 4 is the only new case comparing to the basic alignment. Then the DP algorithm's recurrence relation would be

$$D[i,j] = \max \begin{cases} D[i-1, j-1] + f(S[i], T[j]); \\ D[i-1, j] + f(S[i], \text{-}); \\ D[i, j-1] + f(\text{-}, T[j]); \\ 0 \end{cases}$$

Then the $D[m, n]$ will be the last cell in the DP table: optimal suffix alignment between S and T. Previously, initial cases might have negative scores. Due to the new rule here, we just put zeros.

Consider a suffix alignment example where match = 1, mismatch = indel = -1.

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

This gives alignment $\begin{matrix} & \texttt{ATTGA}\\ \texttt{C} & \texttt{ATTC-}\end{matrix}$

3.2 Local Alignment

Recall that for suffix alignment, $D[i, j]$ denote the optimal "suffix alignment" alignment score of $S[1..i], T[1..j]$. I.e., $D[i, j]$ is the maximum alignment score for $S[i'..i]$ and $T[j'..j]$ for all i' and j'. Therefore, optimal local alignment score is just $\max_{i,j} D[i, j]$. The algorithm will be straightforward:

Algorithm 3: Local alignment

1 Fill the dynamic programming table is the same as suffix alignment.
2 Find (i, j) to maximize $D[i, j]$, and backtrack from there.

For example,

		C	A	T	T	C
	0	0	0	0	0	0
A	0	0	1	0	0	0
T	0	0	0	2	1	0
T	0	0	0	1	(3)	0
G	0	0	0	0	2	2
A	0	0	0	0	1	1

Then the local optimal alignment is the optimal suffix alignment of $T[1..4]$ and $S[1..3]$.

The algorithm was first proposed by Temple Smith and Michael Waterman in 1981. It works for both linear and affined gap penalty. It is known popularly as the Smith-Waterman algorithm. The global alignment algorithm was called the Needleman-Wunsch algorithm, which was published in 1970.

Time complexity is quadratic. Space complexity is $O(mn)$. If we only want the score, we can just use a max variable, which takes $O(1)$ space. If we want the end positions, namely i, j, we can still introduce extra two variables $\max I, \max J$.

If we want the start positions, namely i', j', it would be a bit harder. Recall in suffix alignment, there are four cases. For case 4, we have $i' = i + 1, j' = j + 1$. For other three cases, we simply copy i', j' from the smaller alignment (sub-alignment) because they are same. So we just introduce two variables $\max I', \max J'$.

Similarly, we can do affine gap local alignment:

$$D_0[i,j] = f(S[i],T[j]) + \max \begin{cases} D_0[i-1,j-1]; \\ D_1[i-1,j-1]; \\ D_2[i-1,j-1]; \\ 0 \end{cases}$$

$$D_1[i,j] = \text{gapext} + \max \begin{cases} D_0[i,j-1] + \text{gapopen}; \\ D_1[i,j-1]; \\ D_2[i,j-1] + \text{gapopen}; \\ 0 \end{cases}$$

$$D_2[i,j] = \text{gapext} + \max \begin{cases} D_0[i-1,j] + \text{gapopen}; \\ D_1[i-1,j] + \text{gapopen}; \\ D_2[i-1,j]; \\ 0 \end{cases}$$

Algorithm is as before, except that score is now lower bounded by 0. Afterward, find maximum element in all 3 tables, and backtrack until reaching a 0.

3.3 Many local alignments

It's sometimes useful to find many local alignments of S and T. For example, when there are multiple similar regions between the two input strings. We can let the algorithm output multiple alignments.

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

3.4 Fit Alignment

There are some scenarios where local alignment is not best model. Given sequences S and T. Find a global alignment between S and a substring of T, maximizing the alignment score. We are trying to fit S into T. Deleting the prefix of T is free, deleting the suffix of T is free.

We can use similar idea as before. The DP table aligns T horizontally, S vertically. We can start from anywhere from T, so all zeros. Then for S, we need to initialize properly, $-1, -2, \ldots$. For local alignment, we can stop at any i, j. For fit alignment, we can stop at any j but not any i: we need to find the max value in the last row.

3.5 Linear Space Alignment

Why linear space? Computer RAM used to be very expensive in 80s. There was a prediction " The cost for 128 kilobytes of memory will fall below 100 bucks in the near future" Creative Computing magazine. December 1981, page 6. Even today, keeping everything in the L2 cache may speed up the computation. We have learned the linear space if only alignment score, instead of the alignment, is required. Let's now develop a linear space alignment. We focus on **global** alignment model first.

The idea is to use **divide and conquer**. We want to find j such that the optimal alignment between S and T consists of two parts:

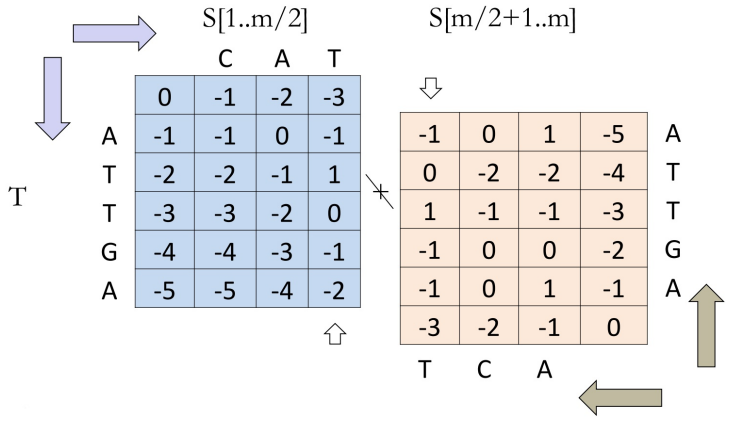
- $S[1..m/2]$ aligns with $T[1..j]$
- $S[m/2+1..m]$ aligns with $T[j+1..n]$

Then we can use divide and conquer. However, we need to compute j in linear space. Note that there may be more than one j satisfying the condition. Any one of them will do the job.

Claim j satisfies the desired condition iff it maximizes

$$\text{alignScore}(S[1..m/2], T[1..j]) + \text{alignScore}(S[m/2+1..m], T[j+1, n])$$

Then we can loop for all j. Let D be the dynamic programming table for aligning S and T. Note that $\text{alignScore}(S[1..m/2], T[1..j])$ is stored in the middle column $(m/2)$ of the table, which takes linear space only. For $\text{alignScore}(S[m/2+1..m], T[j+1, n])$, we can reverse the alignment, which doesn't affect the score. So we can fill DP table backward.



Computing the two center columns requires linear space. Then the algorithm is as follows:

Algorithm 4: Linear Space Alignment

```
1 Align(S, T):
2     if |S| = 1 then
3         return a trivial alignment
4     Use the previous idea to find j that maximizes
        alignScore(S[1..m/2], T[1..j]) + alignScore(S[m/2..m], T[j+1..n])
5     return Concatenation of Align(S[1..m/2], T[1..j]) and Align(S[m/2..m], T[j+1..n])
```

To see its time complexity, we can either use induction or do in in a sloppy way. We know that

$$T(m, n) \leq mn + T(m/2, j) + T(m/2, n-j)$$

Then if we expand the subproblems, the time complexity becomes

$$\begin{aligned} T(m,n) &\leq mn + T(m/2, j) + T(m/2, n-j) \\ &\leq mn + \frac{mn}{2} + \frac{mn}{4} + \cdots \\ &\leq 2mn \end{aligned}$$

But we have linear space complexity.

Related papers:

- *A linear space algorithm for computing longest common subsequences* by D.S. Hirschberg.
- *Optimal alignments in linear space*.

  > "The goal of this paper is to give Hirschberg's idea the visibility it deserves by developing a linear-space version of Gotoh's algorithm."

How to do local alignment in linear space? Recall the trick to find the boundaries of optimal local alignment in linear space. We can use this trick, which takes linear space, to find i, i', j, j'. Then take these four numbers, and then call global linear alignment to get the actual alignment. This is called reduction. Similarly, we can do affined gap penalty in linear space.

4

Score and Significance

Consider the sequence from https://www.ncbi.nlm.nih.gov/protein/6WPT_D:

```
>pdb|6WPT|D Chain D, S309 neutralizing antibody heavy chain
QVQLVQSGAEVKKPGASVKVSCKASGYPFTSYGISWVRQAPGQGLEWMGWISTYNGNTNYAQKFQGRVTM
TTDTSTTTGYMELRRLRSDDTAVYYCARDYTRGAWFGESLIGGFDNWGQGTLVTVSS
```

This is the heavy chain sequence neutralizing antibody in the COVID. If we search this sequence in BLAST, we can find other antibody which has similar sequences, in terms of the scores.

Most optimization problems are of the format:

- **Instance**: describes the input
- **Feasible** Solution: describes the format of the output
- **Score Function**: measures how good a solution is
- **Objective**: either maximize or minimize the score.

For example, in the context of sequence alignment, we have

- Instance: two sequences S and T
- Feasible Solution: insert gaps into S and T so that they have the same length
- Score Function: $\sum_{\text{col}_i} \text{score}(i)$
- Objective: to maximize the score

Consider three purposes of the score function.

- Purpose 1: It helps us to compare solutions of the same instance.

  Which alignment is the best for the same input (s, t).

- Purpose 2: It helps us to compare solutions of different instances.

  Which of (s, t) and (x, y) is more likely to be a homology?

- Purpose 3: It helps us to tell how significant the solution is.

  Does the alignment between s and t indicate that they are homologous? Or if we tell our friend our GPA is 4.0, and they know what that means.

4.1 Purpose 1 of score function

We first examine how the scoring function is designed for the first purpose - compare two alignments and tell which one is better. Recall that what we really want is to find out homologies. Consider two sequences `ATGCATGTA` and `ATGTACTGA`. We want to find their evolution path. Here first `A` doesn't change, while fourth `C` and `T` are different. The better alignment would reflect a higher probability that this alignment happened in the revolution history.

Now considered a simple (oversimplified) evolutionary model. We assume

- evolution only contains substitution and indel.
- two mutations do not overlap. For example, we do not consider the possibility that $A \to C \to A$. This guarantees that all evolutionary information but the order is represented by the alignment.
- different columns are independent to each other.

Along the path of evolution, we denote the probability: p unchanged, q substitution, r indel and $p + q + r = 1$.

Consider the alignment

```
ATGCA-TGTA      (S)
|||  |  ||  |
ATGTACTG-A      (T)
```

we can see that under this model, the probability of the alignment is $p^7 \cdot q \cdot r^2$. It's not convenient to do multiplication in the program, then we convert it into summation. Thus we want to maximize this probability

$$\log(p^7 \cdot q \cdot r^2) = 7 \log p + \log q + 2 \log r$$

Let match $= \log p$, mismatch $= \log q$, indel $= \log r$. We get a scoring scheme. Then maximizing the score is equivalent to maximizing the probability of evolutionary history.

This is sufficient to compare the alignments of the same two sequences. However, there are some problems:

- As probability is always less than 1, log is *always negative*, which is not intuitive.
- Local alignment becomes *meaningless*. The score is always negative, and the empty alignment gives 0 score, which is always the maximum.
- Repeating the alignment twice make the *score lower*. Imagine we have two strings S, T aligned perfectly well. Then consider $S' = S \mid S, T' = T \mid T$. Intuitively, S' and T' produce longer good alignment, but it has lower (more negative) score.

And above example shows that a score works well when comparing the different alignment of the same string, but might be useless when in comparison of two alignments of different pairs of sequences.

For the first problem, if we simply add a minus sign to the score, we are basically change maximization problem to minimization problem, which is the same as before, which doesn't solve the problem.

4.2 Purpose 2 of score function

Given a scheme that match = 1, mismatch = indel = -1, the motivation behinds it is that in homology, we see a lot of matches. This brings us to the idea of **likelihood ratio**.

Consider two contrast models:

- Model 1 (homology): the alignment A between S and T reflects evolutionary history.
- Model 2 (random): the alignment A between S and T is merely a random event.

If homology model gives us higher probability than random model, then we consider it as homology. We want to examine the likelihood ratio:

$$\Pr(\text{alignment} \mid \text{homology}) / \Pr(\text{alignment} \mid \text{random})$$

If it's much bigger than 1 (such as 100000), it's evidence towards model one being the truth. If it's much below 1 (such as 0.00001), it's evidence towards model two being the truth.

Assume for the homology and random models, we have established the probabilities for each column type:

- Match: p and p'
- Substitution: q and q'
- Indel: r and r'

For the alignment

```
ATGCA-TGTA      (S)
|||  |  ||  |
ATGTACTG-A      (T)
```

we have

$$\Pr(\text{alignment} \mid \text{homology}) / \Pr(\text{alignment} \mid \text{random}) = (p/p')^7 \cdot (q/q') \cdot (r/r')^2$$

We usually take a logarithm. The sore becomes

$$7\log(p/p') + \log(q/q') + 2\log(r/r')$$

If this is very positive, then homology model explains the alignment better than random model. And vice versa.

It turns out that this is a better scoring scheme. It prefers column types happens more often in the homology model than in the random model. Usually,

- $p > p'$, therefore a matching column has a positive score
- $q < q'$, therefore a mismatching column has a negative score
- $r < r'$, therefore an indel column has a negative score.

Thus, $\log(p/p') > 0, \log(q/q') < 0$. This avoids the problems we had when only probabilities (not the ratio) were used. In the previous "failed" case, if we put two positive alignments together, we increase the homology chance. Moreover, it can be not only used to compare the alignments of the same input, but also compare the alignments of different inputs, or different lengths. This is indeed the scoring scheme we have seen and have used in practice (in BLAST etc.)

The probability values used in the homology and random models may be obtained by simple counting their frequencies in some "real" alignments and "random" alignments, respectively. Often, the statistics is only approximate and does not need to be precise. In particular, the "random" model often uses some (over)- simplified values. For example: $\Pr(indel) = 0.2, \Pr(match) = 1/20 \times 0.8, \Pr(mismatch) = 19/20 \times 0.8$.

We can still refine this statistics model. Some substitutions are between letters that have similar properties, which then happen more often. The non-indel columns can be further refined to have different scores for different pairs of letters.

For each pair of letters a and b, assume the probability of seeing (a, b) in a column is

$$p(a, b) = \Pr(a, b \mid \text{homology})$$

for the homology model, and is

$$q(a, b) = \Pr(a, b \mid \text{random})$$

for the random model. Then substitution score is then $\log(p(a, b)/q(a, b))$. This is called a substitution matrix.

Let us assume that $q(a, b) = p(a)p(b)$, i.e., independent and random. Here $p(a)$ is the frequency of letter a in the sequences. Note that this is an (over)-simplification. But it provides "good enough" values in practice.

The substitution matrix is particularly important when aligning protein sequences because there are 20 amino acids, some of them share significant similarities and protein alignments have fewer matching columns.

```
Conserved domain database 22426:
KOG4652, HORMA domain [Chromatin structure and dynamics]

Conserved domain length = 324 residues, 100% aligned   ungapped alignment

CT46          15  VFPNKISTEHQSLVLVKRLLAVSVSCITYLRGIFPECAYGTRYLDE--DLCVKILREDKNCPG--STQLVKWMLGC
                    PN +  E QSL  + RLL V++S I   RGIFPE  +  RY+D    L + +LR      G   + L K +
KOG4652       1   TLPNGLENEKQSLEFMTRLLYVAISTILRERGIFPEEYFKDRYVDGNLLVMTLLRRQDAPEGRLVSWLEKGV---

CT46          85  YDALQKKYLRMVVLAVYTNPEDPQTISECYQFKFKYTNNGPLMDFISKN------QSNESSMLSTD-TKKASILL
                  +DA+++K L+ + L V T  EDP+ I E Y F F Y   G +   I+        ++ E S LS D T++    L
KOG4652       73  HDAIRQKLLKKLSL-VITESEDPEDI-EVYIFSFVYDEEGSVSARINYGINGQSSKAFELSQLSMDDTRRQFAKL

CT46         154  IRKIYILMQNLGPLPNDVCLTMKLFYYDEVTPPDYQPPGFKDGDCEGVIFEGEPMYLNVGEVSTPFHIFKVKVTT
                  IRK++I  Q L PLP    +     YY E  PPDYQP GFKD          P  +N+G VSTP H   VKV
KOG4652      146  IRKLHICTQLLEPLPQ-GLILSMRLYYTERVPPDYQPEGFKDSTRAFYTLPVNPEQINIGAVSTPHHKGFVKVL-

CT46         229  ERERMENIDSTILSPKQIKTPFQKILRDKDVEDEQEHYTSDDLDIETKMEEQEKNPASSELEEPSLVCEEDEIMR
                                                  SD  D   K E
KOG4652      219  ----------------------------------------SDATDSMEKAER----------------------T

CT46         304  SKESPDLSISHSQVEQLVNKTSELDMSESKTRSGKVFQNKMANGNQPVKSSKENRKRSQHESGR---IVLHHFDS
                   K S D       V+Q +NK+ E D S S+ ++   +  N + N   PV  S+E+  +SQ   G          D
KOG4652      232  DKISDDP-FDLILVQQELNKSEEADKSFSQEKTTSITPNVLGNPLVPVDQSEEDLLKSQDSPGTGRCSCECGLDV

CT46         376  SSQESVPKRRKFSEPKEHI
                  S Q SVPK RK     EH
KOG4652      306  SKQASVPKTRKSCRKTEHG


Homology between CT46 and MGC26710 hypothetical protein

Identities = 136/249 (54%), with conservative changes = 180/249 (72%)

CT46          1   MATAQLQR-----TPMSALVFPNKISTEHQSLVLVKRLLAVSVSCITYLRGIFPECAYGTRYLDDLCVKILREDK
                  MATAQL              VFP++I+ EH+SL +VK+L A S+SCITYLRG+FPE +YG R+LDDL +KILREDK
MGC26710      1   MATAQLSHCITIHKASKETVFPSQITNEHESLKMVKKLFATSISCITYLRGLFPESSYGERHLDDLSLKILREDK

CT46         71   NCPGSTQLVKWMLGCYDALQKKYLRMVVLAVYTNPEDPQTISECYQFKFKYTNNGPLMDF--ISKNQSNESSMLS
                   CPGS  +++W+ GC+DAL+K+YLRM VL +YT+P   + ++E YQFKFKYT  G  MDF   S + S ES   +
MGC26710     76   KCPGSLHIIRWIQGCFDALEKRYLRMAVLTLYTDPMGSEKVTEMYQFKFKYTKEGATMDFDSHSSSTSFESGTNN

CT46         144  TDTKKASILLIRKIYILMQNLGPLPNDVCLTMKLFYYDEVTPPDYQPPGFKDG-DCEGVIFEGEPMYLNVGEVST
                   D KKAS+LLIRK+YILMQ+L PLPN+V LTMKL YY+ VTP DYQP GFK+G +   ++F+ EP+ + VG VST
MGC26710     151  EDIKKASVLLIRKLYILMQDLEPLPNNVVLTMKLHYYNAVTPHDYQPLGFKEGVNSHFLLFDKEPINVQVGFVST

CT46         218  PFHIFKVKVTTERERMENIDSTIL  241
                   FH  KVKV TE  ++ ++++ +
MGC26710     226  GFHSMKVKVMTEATKVIDLENNLF  249
```

The first gap is circled in red. Anything before the gap is called *ungapped alignment*. For the purpose of this course, we call it a block. Blocks do not have indel.

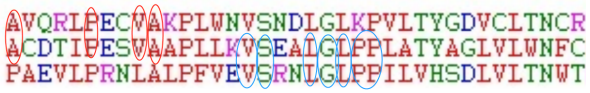
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Consider a substitution matrix shown above: BLOSUM 62. "BLO" stands for block, "SU" for substitution, "M" for matrix. It is the most used amino acid substitution matrix. Note that `B, Z, X, *` are non-standard letters:

1. `B = D or N`
2. `Z = E or Q`
3. `X = any`
4. `* = translation stop`

We can see the red cell in the matrix is also positive even row and col indices are different. Let's study how this is constructed.

Now we want to find out the probability that `A` and `B` appears in the same column of the real homology alignment. We first need to find many real homology examples. Henikoff's published two papers: *Automated assembly of protein blocks for database searching*, *Amino acid substitution matrices from protein blocks*. If we see ungapped alignment between two sequences that have lots of matches, then we consider it as *conserved regions*, then we assume it is real homology.



The identity level is high therefore we know they are homologous without a score matrix. In the two papers mentioned above, there was such a database consisting those blocks.

In the "block" above, there are 37 columns, and each column 3 pairs. Thus in total 111 pairs. For example, the pair `I-L` occurs 3 times; the pair `L-L` occurs 13 times, then

$$P_{IL} = \frac{3}{111}, P_{LL} = \frac{13}{111}$$

There are 111 amino acid in total [1]. There are total 2 I's and 21 L's, thus

$$P_I = \frac{2}{111}, P_L = \frac{21}{111}$$

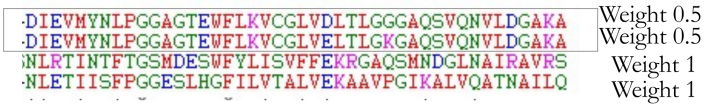
BLOSUM's formula for score are

$$\text{score}(x, y) = 2\log_2 \frac{P_{xy}}{2P_x P_y}, \text{ if } x \neq y$$

$$\text{score}(x, x) = 2\log_2 \frac{P_{xx}}{P_x P_x}$$

In BLOSUM matrices these values are rounded to the nearest integer. The multiplication by 2 at the front provides tiny benefit when rounding to the nearest integer: the absolute round error gets halved, which means the rounding is more accurate.

There's problem with the database. Some blocks might be big, while some might be small. Moreover, there's sample bias: some protein families are more well studied so they are overrepresented in the database. Such bias is caused by the studies, not reflecting what's going on during evolution. To remove this bias in statistics, those "redundant" proteins are classified together before BLOSUM calculation.



Consider the first two sequences in the picture. We can see they are overly similar. The sequences that are 62% or above similarity are grouped together and given total weight 1. This way, the `AA` pairs are counted between groups that are 62% similar or below. The lower this number is, the better is the matrix suitable to distant homology search. The original BLOSUM paper found out 62 is best at the time the paper was prepared.

4.3 Purpose 3 of score function

Consider that BLAST matches a query sequence with all database sequences, and return the highest scoring local alignments. The best local alignment score is 100. Does it mean good or bad? The answer depends on the score scheme you use so we need to standardize it for effective communication. Intuitively, we would ask "can this happen randomly"? This is the intuition for the significance. This is formalized as the p-value.

> *p-value*
>
> Imagine we want to prove that a coin is biased on its two sides. We first create *null hypothesis* H_0: the coin is fair. Or *alternative hypothesis*: the coin is biased, which is what we try to prove. Then we conduct the experiment: draw the coin 20 times, and found 14 heads and 6 tails.
>
> $$\Pr(\#\text{ heads } \geq 14 \text{ or } \#\text{ tails } \geq 14 \mid H_0) = 0.1154$$
>
> This is called the p-value.
>
> In statistical hypothesis testing, the p-value is the probability of obtaining a result at least as extreme as the one that was actually observed, given that the null hypothesis is true. A small p-value **rejects** the null hypothesis. So, we choose to believe the alternative hypothesis.
>
> In the coin example, 0.1154 is certainly not good enough. But what if we draw 40 times, and found 28 heads and 12 tails? Now p-value is 0.0115. This is how scientists show the effectiveness of a treatment through clinical trials. (null hypothesis: it is not effective).

[1] a this is coincident, not the same as 111 pairs before. It depends of number of sequences in each block.

> A small p-value *rejects* the null hypothesis. So, we choose to *believe* the alternative hypothesis. This does not mean that the alternative hypothesis is surely correct. It is just the null hypothesis is unlikely to be correct. It is just a way to communicate the significance. The union of null hypothesis and alternative hypothesis is not the whole universe. For example, if the two sequences are irrelevant, then we should not have seen the alignment with such a high score. But they may be related for reasons other than homology - e.g. convergence evolution.
>
> In practice an arbitrary threshold 0.05 is often used, for no apparent reason.

Now we can use p-value for local alignment to communicate its significance. Null hypothesis: *the query S and the database T are irrelevant*. Now we observe a high scoring local alignment with score x. This is an extreme case. Then the p-value is $\Pr(\text{score} \geq x \mid H_0)$. How to compute the p-value. From now on, S and T are long sequences.

Statistics of Ungapped Local Alignment

First bioinformatics studied statistics form ungapped local alignment. Without indel, things become easier. For ungapped alignment (no indel), each local ungapped alignment is called an **HSP** (High Scoring (Segment) Pair) in the BLAST program. For an individual HSP, the alignment score is the sum of n identical independent variables. So the score is a binomial distribution. As $|S| = n, |T| = m$ are quite long, there are about $O(mn)$ HSPs. As we have a lot of such ungapped alignments, and we only output the best one (with the highest alignment score). For p-value, we need to check how *rare* this best alignment has a high score. The maxima of many identical independent variables is a so-called extreme value distribution.

There are two papers in 90s studying the distribution of ungapped local alignment (HSPs) among two big sequences via both E-values and P-values: "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes" and "Limit distribution of maximal nonaligned two-sequence segmental score".

Theorem 4.1: E-value

In the limit of sufficiently large sequence lengths m and n, the statistics of HSP scores are characterized by two parameters, K and λ. Most simply, the expected number of HSPs with score at least x is given by the formula

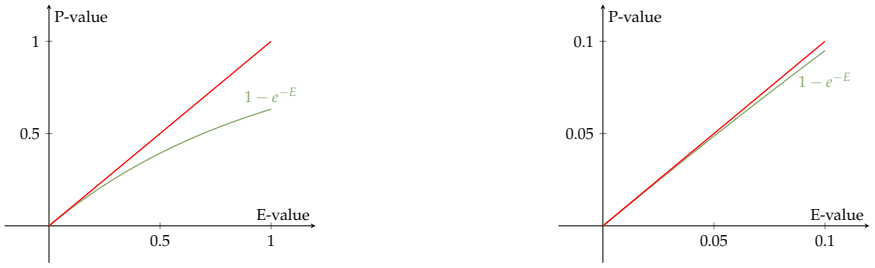
$$E = Kmne^{\lambda x}$$

This is called the E-value of the alignment. The parameters K and λ can be somehow determined from the scoring matrix. (Details not given here). Thus, we just convert the alignment score x to E-value. If it is very small, then random sequences will not often produce HSPs with score $\geq x$. This is likely caused by homology. There are not many false positives if we treat every local alignment with score $\geq x$ as homologies.

Theorem 4.2: P-value

The number of random HSPs with score $\geq S$ is described by a Poisson distribution. Therefore the chance of finding zero HSP with score $\geq S$ is e^{-E}, where E is the E-value of score S.

When E is small, then $e^{-E} \to 1$, and probability of finding no HSP with that score is high. Therefore, the p-value is $1 - e^{-E}$. When E-value is very small, P-value and E-value are almost identical. BLAST chose to use E-value. Below are comparisons of two values.



Green curve is p-value, which is equal to $1 - e^{-E}$. We can see from the diagram on the right: when E-value is between 0 and 0.1, the green curve matches the red curve nicely. All we care are the cases where both are small. When both are big, it's hard to see its homology.

The E-value and P-value have their physical meanings. For example, if S and T are random, we expect to see on average 0.01 HSP with such a high score. The alignment score, however, largely depends on the scoring matrix one chose. Saying that "the alignment score is 100" is like saying "the length is 100". There are *no* units for this measurement.

The statistics of gapped alignments The statistics developed above have a solid theoretical foundation only for local alignments that are not permitted to have gaps. However, many computational experiments and some analytic results strongly suggest that the same theory applies as well to gapped alignments. But we need to know how to compute K and λ. We don't know how to compute/estimate them mathematically. Thus, we need to do some simulation like repeating random alignments.

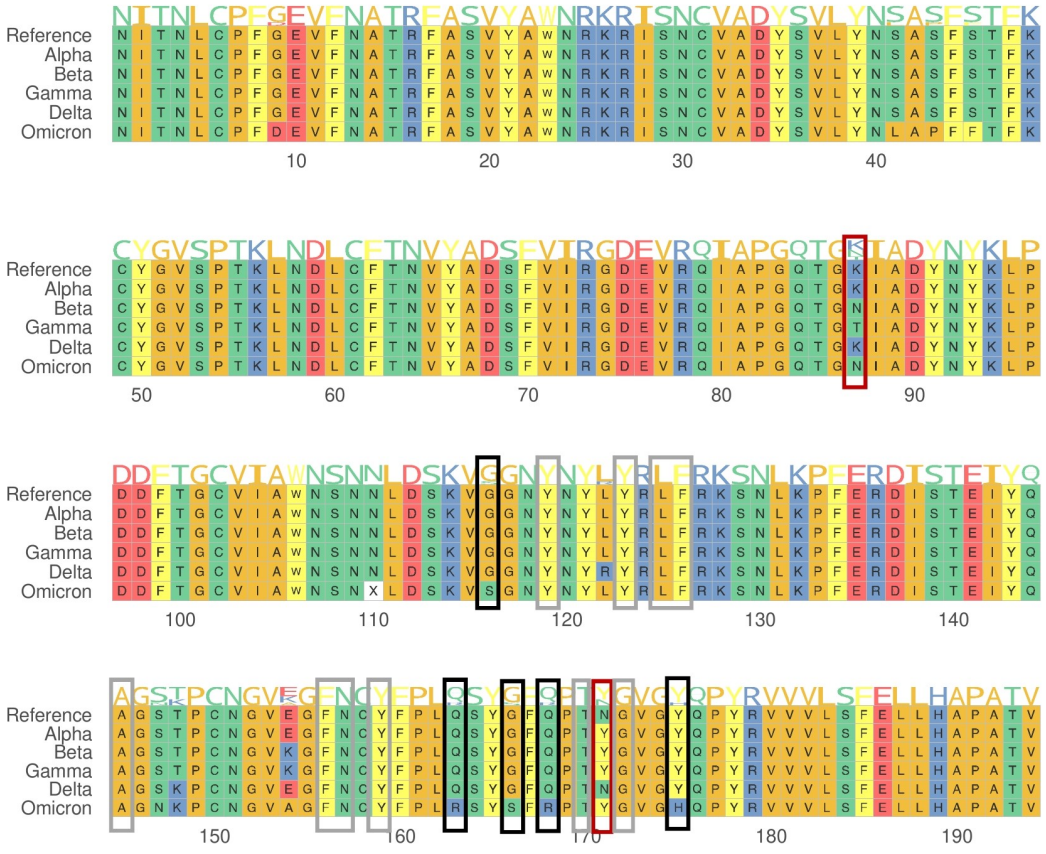
5

Multiple Sequence Alignment

Consider an example of multiple sequence alignment.

bioRxiv preprint doi: https://doi.org/10.1101/2021.12.08.471688; this version posted December 9, 2021. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY-NC-ND 4.0 International license.

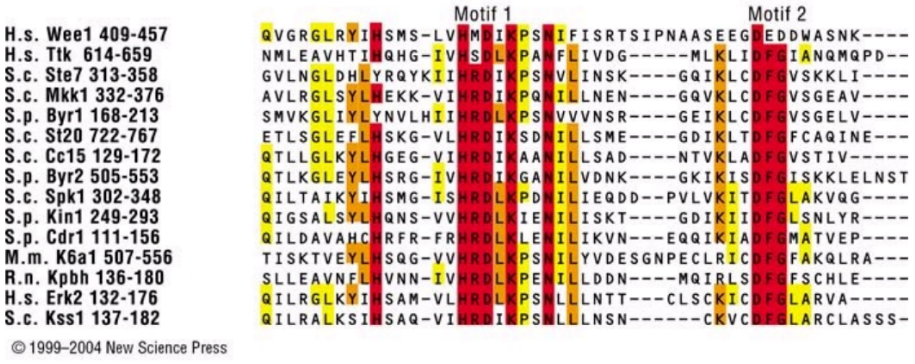
Fig.3



From this paper:

> *Contacting residues between SARS-CoV-2 and ACE2.* Boxes denote the contacting residues. Black boxes denote mutations unique to omicron, red boxes denote mutations occurring in multiple variants, and grey boxes denote no mutations in any variant.

If we consider indels, alignment becomes harder. A multiple sequence alignment of k sequences is an insertion of gaps in the positions of the sequences, just like a pairwise alignment.



Red ones and yellow ones are almost the same. Arthur M. Lesk once said:

> "Two homologous sequences whisper, a multiple alignment shouts loudly."

5.1 Heuristic Algorithm for Multiple Alignment

There is a simple algorithm that can merge pairwise alignments to a multiple alignment. The algorithm does not guarantee the optimality of the result. But runs relatively fast.

Suppose we have the following two pairwise alignments:

- t: `A-GAGC`
- s_1: `ATGAGC`
- and
- t: `A-GAGC`
- s_2: `AGTTGC`

Our main idea is to use the shared sequence t to construct a multiple alignment of s_1, t, and s_2.

```
 t: A-GAGC              t: A-GA-GC
s1: ATGAGC      ⟹     s1: ATGA-GC      ⟹      t: A-GA-GC
    and                    and                s1: ATGA-GC
 t: AGA-GC              t: A-GA-GC             s2: A-GTTGC
s2: AGTTGC             s2: A-GTTGC
```

We observe that pairwise alignment is "induced" by the multiple alignment.

Algorithm Insert gaps to the two alignments, so that the superstrings for t become the same in the two alignments. Then put the two alignments together. We keep two pointers i and j for t in both pairs.

Property Maintains the pairwise alignment unchanged if ignoring the all-gap columns of the pairwise alignment.

The same idea can be applied to merging two multiple alignments as well. For two multiple alignments, as long as they share a common string, we can apply the same idea.

```
 t: A-GA-GC              t: A-GA-GC
s1: ATGA-GC             s1: ATGA-GC              t: A-GA-GC
s2: A-GTTGC    ⟹       s2: A-GTTGC      ⟹      s1: ATGA-GC
                                                 s2: A-GTTGC
 t: AGAGC                t: A-GA-GC              s3: A-TA--C
s3: ATA-C               s3: A-TA--C
```

The process can be continued to merge two multiple alignments together as a bigger one. If we always use t is the common string, the alignment between other sequences and t have not changed over the iterations. Note that the obtained multiple alignment may not be optimal. In the example above, if we only consider the pair alignment between s_1 and s_3, we should only open the gap between the second `T` and third `A`.

Algorithm 5: Heuristic Algorithm for Multiple Alignment

Input: $s_1, \ldots, s_n$

$A \leftarrow$ pairwise alignment of s_1 and s_2.
for $i \leftarrow 3..n$ **do**
  Construct pairwise alignment P between s_1 and s_i.
  $A \leftarrow \text{merge}(A, P, s_1)$, i.e., merging A and P using s_1 as the template.
return A

The order of the merging is important to get good (but not optimal) multiple alignment. We want to merge similar sequences first, as we don't want to open gaps in the beginning, which will be carried over for future merging operations. One way is to construct a minimum spanning tree, and then merge using the shared vertices.

The exact algorithm for multiple alignment is super-polynomial. Before we study it, let's examine a heuristic algorithm. A heuristic algorithm is an algorithm that gives up quality for speed. It usually does not offer any performance guarantee in terms of quality. Well... We already sacrificed the quality because of a simple scoring function anyway. If we cannot afford exponential time, the best we can ask for is a suboptimal solution. But practically, it might work sometimes. We do not want to spend super-polynomial (e.g. exponential) time.

5.2 Exact Algorithm for Multiple Alignment

When the optimal alignment is needed. There is an exact algorithm as well. First we need to define **score function** we want to optimize.

For each column, assuming the letters are $a_1, \ldots, a_n$ for the n sequences. Define a column score $S(a_1, \ldots, a_n)$. Then the alignment score is the total of the column score. There are more than one ways to define the column score. Let us examine the simplest one first.

SP-score

SP (Sum of Pair) score is the most widely used because its simplicity. We have a similarity matrix $s(a, b)$ for any two given letters (or dashes). For the k-th column with n letters $x_1, \ldots, x_n$. Define the SP-score for the k-th column by

$$S_k = \sum_{i,j} s\left(x_i, x_j\right)$$

The SP-score of the alignment is

$$\sum_{1 \le k \le m} S_k$$

We normally require $s(\text{-},\text{-}) = 0$.

Note that SP-score can be viewed from a different perspective as follows: For each pair of sequences S_i and S_j (including the gaps) in the multiple sequence alignment, define

$$S_{i,j} = \sum_k (S_i[k], S_j[k])$$

The SP score is then $\sum_{i,j} S_{i,j}$. The two perspectives are equivalent if $s(\text{-},\text{-}) = 0$.

Recall the pairwise alignment algorithm, we then borrow this idea to align three sequences. Use `X` to indicate a letter. The last column has 7 cases.

X	-	X	-	X	-	X
X	X	-	-	X	X	-
X	X	X	X	-	-	-

Let f be the column-wise score function. $DP[i_1, i_2, i_3]$ is the optimal score for $s_1[1..i_1], s_2[i..i_2], s_3[1..i_3]$. $DP[i_1, i_2, i_3]$ is maximum of the 7 cases.

- case `XXX`: $DP\left[i_1, i_2, i_3\right] = DP\left[i_1 - 1, i_2 - 1, i_3 - 1\right] + S\left(s_1\left[i_1\right], s_2\left[i_2\right], s_3\left[i_3\right]\right)$
- case `XX-`: $DP\left[i_1, i_2, i_3\right] = DP\left[i_1 - 1, i_2 - 1, i_3\right] + S\left(s_1\left[i_1\right], s_2\left[i_2\right], \text{-}\right)$
- ...

It is necessary to introduce some notation for n sequences...

Here `X` indicates a letter and a dash indicates an indel. If you regard `X` as 1 and - as 0. Then these cases are all the 3-bit binary numbers but `000`. In general, for n sequences, there are $2^n - 1$ cases. Let us use δ_j to indicate whether at sequence j the last column is a letter. $\delta_j = 1$ if it is a letter; $\delta_j = 0$ if it is a dash.

Algorithm 6: Algorithm for Optimal Multiple Alignment

1 Let $S(x_1, \ldots, x_n)$ be a function to compute the column score.
2 Let $D[i_1, \ldots, i_n]$ be the optimal multiple alignment score of $s_1[1..i_1], \ldots, s_n[1..i_n]$.
3 Consider the last column of the optimal multiple alignment. There are $2^n - 1$ cases.
4 For each case, the "sub-alignment" after removing the last column is also optimal.
5 Then

$$D\left[i_1, i_2, \ldots, i_n\right] = \max_{\delta_j = 0,1} \left(D\left[i_1 - \delta_1, \ldots, i_n - \delta_n\right] + S\left(b_1, \ldots, b_n\right)\right)$$

where

$$b_j = \begin{cases} \text{-}, & \text{if } \delta_j = 0 \\ s_j\left[i_j\right], & \text{if } \delta_j = 1 \end{cases}$$

We see that line 5 has a nested loop m^n iterations. For each recurrence relation, we need to maximize for every possible δ_j, 2^n possible cases. Thus the time complexity is $O(m^n 2^n T)$, where T is the time needed for computing a column score. Space complexity is $O(m^n)$.

This problem is NP-hard. So, no polynomial time algorithm exists (unless $P = NP$). Computing the optimal alignment with large m and n is hopeless. But at least this works for small m and n.

SP score (either maximization or minimization) is one of the simplest scoring function in use. There are better proposals to the scoring function. Let's examine a more sophisticated score called relative entropy. We want to maximize.

Relative Entropy

There are n letters $x_1, \ldots, x_n$ in a column. For letter a in alphabet, let n_a be the number of a in the n letters. Our two different models assume two different distributions of letters in that column.

- Model 1: (homology model) a occurs with probability $p_a = n_a/n$.
- Model 2: (random model) a occurs with "background" probability q_a, which is learned from a database.

We want to compute the likelihood ratio. Then we do log likelihood ratio as usual. First find the conditional probabilities,

$$\Pr(x_1, \ldots, x_n \mid \text{model } 1) = \prod_{1 \le i \le n} p_{x_i}$$
$$\Pr(x_1, \ldots, x_n \mid \text{model } 2) = \prod_{1 \le i \le n} q_{x_i}$$

Then log likelihood ratio is

$$\log \prod_{1 \le i \le n} \frac{p_{x_i}}{q_{x_i}} = \sum_{1 \le i \le n} \log \frac{p_{x_i}}{q_{x_i}} = \sum_{a \in \Sigma} n_a \log \frac{p_a}{q_a}$$

This is called the relative entropy at a column. Let E_j be the relative entropy of column j. Then the alignment score is equal to $\sum_{1 \le j \le m} E_j$, which we want to maximize. Note that $E_j \ge 0$, and $E_j = 0$ only if $p_a = q_a$ (This is because of a theorem in information theory).

A few assumptions of relative entropy score:

- Sequences are independent to each other.
- Columns are independent to each other.

These seem to be too strong but at least when these assumptions are true the score has a theory foundation. Other scores are more empirical. Note: If all q_a are equal to each other, then this is equivalent to the "minimum entropy" introduced in the reference book (Durbin page 138).

5.3 Approximation Algorithm

In the heuristic algorithm, it would be good if we can get some guarantee on the quality of the suboptimal result. Suppose we want to maximize a score, and the optimal value is OPT, how about computing a solution with score at least OPT/c for a small constant $c > 1$? If $c = 1.001$, this is not so bad. If this can be done, this is called a **ratio-c approximation algorithm** for the problem. For minimization problem, ratio c means the algorithm gives score at most $c \cdot OPT$.

The known approximation algorithm for multiple alignment only works for the minimization version of the problem. Let $M[a, b]$ be a distance metric between two letters (possibly -) a and b satisfying **triangular inequality**: $M[a, a] = 0, M[a, b] \ge 0, M[a, c] \le M[a, b] + M[b, c]$.

The distance between two sequences $d'(S_i, S_j)$ is the total of their column scores using M as the score scheme. The SP score is $\sum_{i,j} d'(S_i, S_j)$. We want to minimize. Note that this is almost the same as before, except that now we want to minimize the difference.

For sequences S, T, R taken from the same multiple alignment, we have

$$d'(S, R) \le d'(S, T) + d'(T, R)$$

Note that S, T, R may have the '-' symbols in the sequence. Here are some notations:

- When context is clear (the multiple alignment is given), we also use $d'(s,t)$ to denote the distance between the original sequences s and t (without '-'). It is equal to $d'(S,T)$.
- When there are more than one multiple alignments $A_1, A_2, \ldots, A_n$, we use $d'_i(s,t)$ to indicate the distance induced by A_i.
- We use $d(s,t)$ to denote the optimal pairwise alignment between s and t. Thus $d(s,t) \leq d'(s,t)$.

Algorithm 7: Approximation algorithm for multiple alignment

Input: $s_1, \ldots, s_n$

1 **foreach** $i = 1..n$ **do**
2 Build optimal pairwise alignment between s_i and s_j for each $j = 1, 2, \ldots, n$.
3 Use these pairwise alignment to build a multiple alignment A_i.
4 Output the A_i with the minimum SP score.

We want to prove that this algorithm has ratio 2.

Fact 1

$d'_i(s_i, s_j) = d(s_i, s_j)$ for every j.

Recall that in A_i, s_i is in the center. Thus the induced pairwise alignment is optimal. And $d'_i(s_k, s_\ell)$ might not be optimal if $i \neq k$.

We want an upper bound on the score produced by the algorithm:

$$d'_i(s_k, s_\ell) \leq d'_i(s_i, s_k) + d'_i(s_i, s_\ell)$$

Let A^* be the optimal multiple alignment and d^* be pairwise alignment score induced by optimal multiple alignment A^*. Then we have

$$\begin{aligned}
\sum_{i=1}^{n} SP(A_i) &= \sum_i \sum_{k,\ell} d'_i(s_k, s_\ell) \\
&\leq \sum_i \sum_{k,\ell} \left(d'_i(s_i, s_k) + d'_i(s_i, s_\ell) \right) \\
&= \sum_i \Big(\sum_{k,\ell} d'_i(s_i, s_k) + \sum_{k,\ell} d'_i(s_i, s_\ell) \Big) \\
&= \sum_i \Big(n \cdot \sum_k d'_i(s_i, s_k) + n \cdot \sum_\ell d'_i(s_i, s_\ell) \Big) \\
&= \sum_i 2n \cdot \sum_k d'_i(s_i, s_k) \\
&= \sum_i 2n \cdot \sum_k d(s_i, s_k) \\
&\leq \sum_i 2n \cdot \sum_k d^*(s_i, s_k) \\
&= 2n \cdot \Big(\sum_i \sum_k d^*(s_i, s_k) \Big) \\
&= 2n \cdot SP(A^*) \\
&= 2n \cdot OPT
\end{aligned}$$

So, at least one of the A_i has score no more than $2 \cdot OPT$.

Theorem 5.1

The above algorithm is a ratio-2 approximation to multiple alignment under SP score.

For general score scheme (general score matrix) this ratio-2 may not hold. For example, when the score scheme does not satisfy triangular inequality. Or when the score scheme has both positive and negative scores.

To get a practical solution, the algorithm might combine different ways. Clustal ω, is the most frequently used multiple sequence alignments. Moreover, multiple sequence alignment has many applications:

- **Motif finding**: conserved regions of a protein family may be important motifs.
- **Structure prediction**: after multiple alignment of a protein family, their structures can be predicted together. DeepMind, AlphaFold 2.
- **Phylogeny**: the pairwise alignment induced by the multiple alignment can be more accurate than the optimal pairwise alignment in reality. Build multiple alignment first then do phylogeny by examining the evolution on each column.

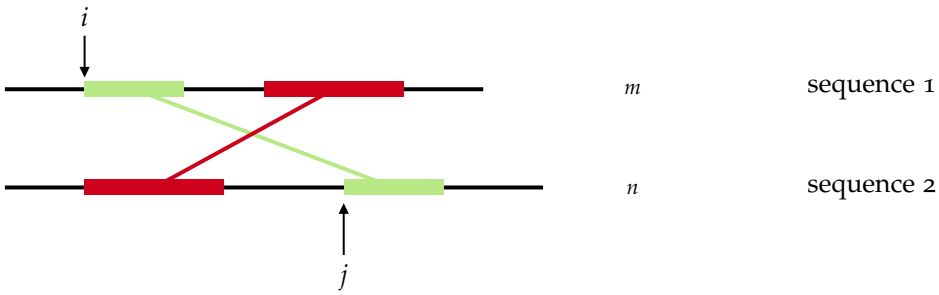
6

Seeding Methods in Homology Search

Recall that BLAST is very powerful. We will study some of their algorithms.

Consider another example. We want to compare two genomes. Smith-Waterman is the most accurate method: do dynamic programming, find high scoring entry and find good local alignment. Time complexity is $O(mn)$. So for human v.s. mouse: $3 \times 10^9 \times 3 \times 10^9 = 9 \times 10^{18}$ which will take many years.

Observe that in most cases, similarities or local alignments are not long, very short relative to the genomes.



For every pairs of (i, j), build a local alignment around it. This takes $O(mnT)$ where T is time for sub-area. This is not better than Smith-Waterman. But this leads to an important idea.

Most pairs of (i, j) are useless. We don't need to compare every i and j. Instead, We only want to try local alignments on the "promising" pairs of (i, j). In the context of sequence similarity search in bioinformatics, these "promising" pairs are called "**seeds**" or "**hits**". We then need a proper definition for hits, and • some efficient way to enumerate the hits faster than trying every pair of (i, j).

6.1 Short Consecutive Match

BLAST uses this idea as hits. The idea: find a consecutive (exact) match, and use it as hit. BLAST uses length 11 exact matches. For random i, j, the probability of exactly length k match between lengths m and n (random) sequences is $(4^{-k} \cdot mn)T$, where T is time spent on each of them. The time complexity becomes $O(4^{-k}mnT)$. By default, BLAST used $k = 11$, and it speeds up about four million.

The Idea behind Seeding

A true similarity has a high chance of being hit. Hitting at any position in the similarity will do. A random pair (i, j) has low chance of being hit. Thus, if we use hit to filter(i, j), we will detect most true similarities, and not wasting time on random pairs of (i, j).

6.2 Data Structure for Finding Hit

We will discuss a simple data structure using index table, which is different from what BLAST is using.

From wiki,

> Usually, the term k-mer refers to all of a sequence's subsequences of length , such that the sequence AGAT would have four monomers (A, G, A, and T), three 2-mers (AG, GA, AT), two 3-mers (AGA and GAT) and one 4-mer (AGAT). More generally, a sequence of length will have k-mers and total possible k-mers, where.

For each k-mer, we build an index table to remember all its occurrences in S. Then for each k-mer of T, find its hits in the index table. The index table can be a trie or a hash table. For example, given $S =$ `AATCTTAA`, we fill the index table as follows:

`AA`	$\longrightarrow$	`0, 6`
`AC`		
`AG`		
`AT`	$\longrightarrow$	`1`
`CA`		
`CC`		
`CG`		
`CT`	$\longrightarrow$	`3`
`GA`		
`GC`		
`GG`		
`GT`		
`TA`	$\longrightarrow$	`5`
`TC`	$\longrightarrow$	`2`
`TG`		
`TT`	$\longrightarrow$	`4`

We see `AA` 2-mer in index 0 and 6 of S. Then we go through 2-mer of $T =$ `GAACTTA`. Thus we found that (index) $j = 2$ (in T) will hit $i = 0, 6$ (in S).

There are 4^k entries. For each entry, it is a list of occurrences of a k-mer in S. All these lists cost $|S|$ space. The space complexity is not bad.

Building the index using S takes $O(n)$. Find matches between the index and sequence T: $O(m)$ time to scan T, plus we need to examine all of the N hits found. Let t be the examination time. Thus the overall runtime is $O(n + m + Nt)$. The term Nt is the most expensive part. Indexing overhead is small. In practice, most of the hits encountered are random hits.

We can do filtration for multiple rounds. In the previous time complexity, t in Nt will be large if the examined sequence is long. We can somehow spend extra time $t' \ll t$, to filter out random hits. Then Nt becomes $N \cdot t' + N' \cdot t$. After finding a hit, instead of trying to build a local alignment directly, BLAST uses another round of filtration to determine if a hit is a "good" or "bad" hit. Quick search in both directions; if most symbols match, it's a good hit. Otherwise it's bad. More precisely, use

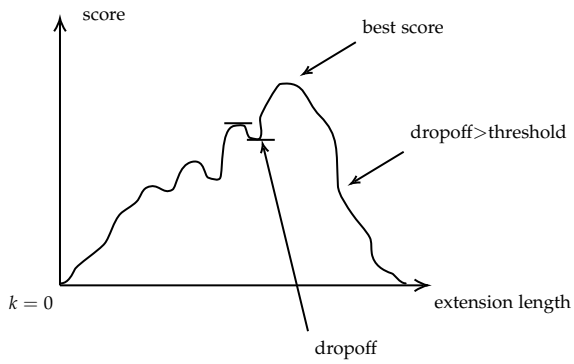
ungapped extension (linear time) to find HSPs. If an HSP is above a certain score threshold, build a local alignment around it.

6.3 HSP extension

Algorithm 8: HSP extension

```
for k ← 0.. do
    score += sc(S[i + k], T[j + k]) // go right
for k ← 1.. do
    score += sc(S[i − k], T[j − k]) // go left
```

But when to stop? Score will increase and decrease during the extension.



Extension stops when drop off greater than threshold.

How long will the extension continue after reaching best score? We assume that

- After reaching best score, sequence becomes random.
  Thus $\Pr(match) = 1/4$ and $\Pr(mismatch) = 3/4$
- match $= 1$ and mismatch $= -1$

Thus expected score on each additional base is -0.5. If $dropoff = k$, then after $2k$ bases, the expected dropoff will reach k.

This idea does not guarantee we will find every single good local alignment. BLAST will fail on the case such that the longest exact match is of length < 11, then BLAST will not find this HSP. This brings us a dilemma:

- **Sensitivity** - needs shorter seeds. the success rate of finding a homology
- **Speed** - needs longer seeds. Mega-BLAST uses seeds of length 28.

6.4 Spaced Seeds

111*1**1*1**11*111 is called a spaced seed. We use * to denote "don't care" positions. For example,

```
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT...
||  |||||||||  |||||  ||  |||||  ||||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT...
       111*1**1*1**11*111
```

Note that * can be match and mismatch: first * is match and second * is mismatch. Hit is all the required matches are satisfied. BLAST's seed is 11111111111.

A homology/similarity region's actual sequences do not matter, the match/mismatch matters. Therefore, a region is often denoted by a binary `0-1` sequence:

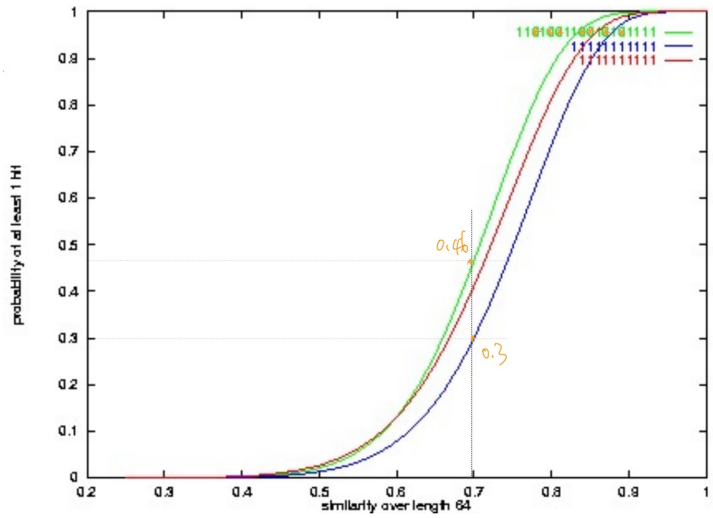
```
110111110011101110111111
```

A hit is then as follows

```
110111110011101110111111
   111*1**1*1**11*111
```

Now we change the definition of hit, we need to change data structure as well, efficiently. We build entries/indexes in the same way as consecutive seed. Except that now we have a length l, weight w seed. E.g. `11*1`. Each l-mer, take the w letters out and put in index table. Then for index `AAA`, it contains list of occurrences of `AA?A` in S. So now we take spaced k-mer, instead of consecutive k-mer. The index table can be a hash table.

The time complexity is the same: still waste the same amount of time on random hits. What about the sensitivity? Below is simulated sensitivity curves. Define HSP of length 64, and randomly assign 1 with probability p, and $p(0) = 1 - p$. x-axis is similarity level.



Why spaced seeds are better? Consider a HSP with similarity level p and a fixed position/index i, then it has match with p probability. If we use weight 11, then $\Pr(\text{seed hit pos } i) = p^{11}$. This doesn't change between spaced seed and consecutive seed. But what we care is $\Pr(\text{HSP contains at least 1 hit})$. But as spaced seeds are longer, the probability is lower. So this explanation doesn't work.

Consider

```
TTGACCTCACC?
||||||||||||?
TTGACCTCACC?
11111111111
 11111111111
```

It seems that we hit twice. However, if we hit pos 0, the chance of hitting pos 1 gets increased: we then only require last position to be matched, of probability p. So the hits between different positions within HSP are correlated. This idea works vice versa. BLAST's seed usually uses more than one hits to detect one homology, which is redundant.

```
CAA?A??A?C??TA?TGG?
|||?|??|?|??||?|||?
CAA?A??A?C??TA?TGG?
111*1**1*1**11*111
 111*1**1*1**11*111
```

For spaced seeds, hits are not strongly correlated, more independent. Spaced seeds uses fewer hits to detect one homology, which is more efficient. We see that PatternHunter's seed do not overlap heavily when shifts:

```
111*1**1*1**11*111
 111*1**1*1**11*111
  111*1**1*1**11*111
   111*1**1*1**11*111
    111*1**1*1**11*111
     111*1**1*1**11*111
      111*1**1*1**11*111
       111*1**1*1**11*111
        ......
```

The hits at different positions are independent. After first hit, the next three shift gives us 6 matches, and so on. Thus the probability of having the second hit is $3p^6 + \cdots$. For BLAST's seed, for the second position, we only need one additional match; for the third position, we need two additional matches. Thus probability is roughly $p + p^2 + p^3 + p^4 + \cdots$, which is bigger than spaced seed.

6.5 Lossless Filtration

Filtration is bad, because we give up sensitivity: some of HSPs will be lost. When seeds are short enough and HSP similarity is high enough, lossless filtration is also possible. For example, seed 111 can guarantee to match when a sufficiently long HSP has similarity 66.7%. To fail being hit by 111, the HSP must have a mismatch in every 3 adjacent positions. On the other hand, $110110110\cdots$, which has 66.6% similarity, will fail the seed 111.

Now consider spaced seed 11*1. We claim that $\forall \epsilon > 0$, seed 11*1 will hit every sufficiently long region with similarity $0.6 + \epsilon$.

Proof:
Suppose there is a sufficiently long region not hit by 11*1. For every HSP, we can divide it into different blocks of the form 1^a0^b. Notice that $a \leq 3$, otherwise it's already hit by 11*1. We then consider different cases for a:

- $a = 3$. We know next block starts with at least one 1. So in order to have mismatch, $b \geq 2$. This has identity level $\leq \frac{3}{5}$ as it has 3 matches out of 5: $\frac{a}{a+b}$.
- $a = 2$. Similarly, $b \geq 2$. Otherwise $11011^{a'}0^{b'}$ will produce a match. Identity level $\leq \frac{2}{4}$.
- $a = 1$. We need to have $b \geq 1$ to form a block. Identity level $\leq \frac{1}{2}$.

Thus in each block, similarity ≤ 0.6. So the long region's similarity is $< 0.6 + \epsilon$. □

6.6 Compute Seed's Sensitivity

We denote a probabilistic distribution of HSP by R, and $\Pr(R[i] = 1) = p$ is uniform random distribution. We want to compute $\Pr(\text{length-}n\ R \text{ is hit by a seed } x)$, where $|x| = k$.

Denote a length-k binary string by s, and Rs to be the concatenation of R and s. Let $D[i, s]$ be the

probability Rs is hit by x for a length-i R. Then by total probability law,

$$\begin{aligned}\Pr(\text{length-}n\ R \text{ is hit by a seed } x) &= \sum_{s' \in \{0,1\}^k} \Pr(s') \cdot \Pr(R_{n-k}s' \text{ is hit by } x) \\ &= \sum_{s' \in \{0,1\}^k} p^{\#1 \text{ in } s'}(1-p)^{\#0 \text{ in } s'} D[n-k, s']\end{aligned}$$

Then how to compute $D[i, s]$? Dynamic programming.

- Case I: s is hit by x. Then $D[i, s] = 1$.
- Case II: s is not hit by x. We have two subcases: R ends with 1 with probability p, and 0 with probability $1 - p$. Denote s' the length-$(k-1)$ prefix of s. Thus we hav the following recurrence relation:

$$D[i, s] = p \cdot D\left[i - 1, 1s'\right] + (1-p) \cdot D\left[i - 1, 0s'\right]$$

Algorithm 9: Seed x's sensitivity

```
Initialize D[0, s] // either 0 or 1, depending on whether s is hit by x
for i ← 1..n do
    for s ∈ {0,1}^k do
        if s is hit by x then
            D[i, s] = 1
        else
            D[i, s] = p · D[i − 1, 1s'] + (1 − p) · D[i − 1, 0s']
return Σ_s Pr(s) · D[n − k, s]
```

Here $\Pr(s) = p^{\#1 \text{ in } s'}(1-p)^{\#0 \text{ in } s'}$. Time complexity is $O(2^k n)$. There exists a more efficient algorithm $O(2^{\#0 \text{ in } x} n)$, which is beyond the scope of this course.

There's no good algorithm to directly optimize the spaced seed, but we can do exhaustive search to enumerate all spaced seeds with weight 11 and no longer than 18, calculate the sensitivity of each, and output the one with the highest sensitivity. This is the ONLY known algorithm that guarantees the finding of optimal seed. Many heuristics exist to find suboptimal seeds.

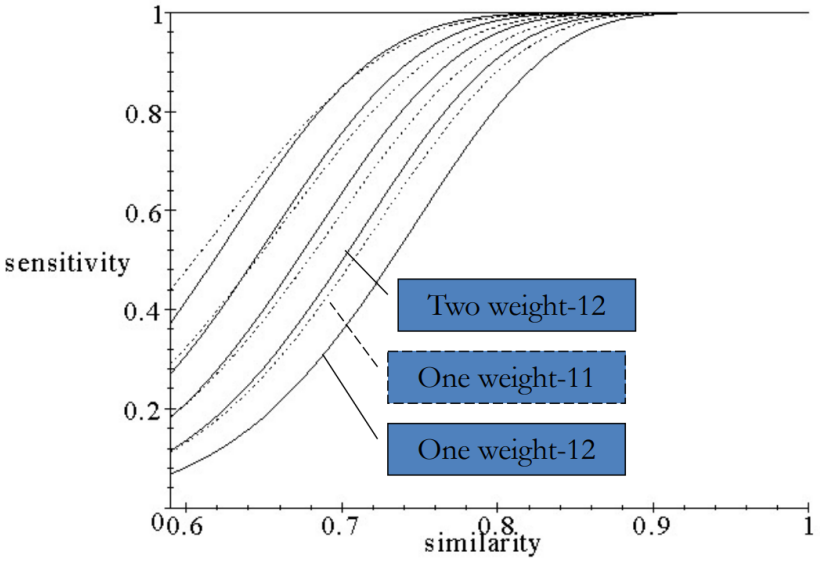
6.7 Multiple Spaced Seeds

Seeds with different shapes can detect different homologies. Some seeds may detect more homologies than others. This leads to the use of optimized spaced seed. Can use several seeds simultaneously to hit more homologies. Approaching 100% sensitive homology search. Also the overlap of seeds is waste of time.

Consider a set of seeds found by optimization algorithm: (homology identity = 0.7, homology length = 64)

```
111*11**1*11*1*111
1111***1***1**11*1*111
11**11*1**1*1***11*111
111*1***1111**1***11*1
```

To use multiple seeds, one only needs to search multiple times with different seeds, and combine results. Of course, you can search with them simultaneously. In either case, this slows down approximately k times if k seeds are used. Is it worth it? How does it compare with using one shorter seed? This is a trade-off. We can consider the tradeoff between weight of seeds and number of multiple seeds.



Solid curves: Multiple $(1, 2, 4, 8, 16)$ weight-12 spaced seeds. Dashed curves: Optimal spaced seeds with weight $= 11, 10, 9, 8$. Typically, "Doubling the seed number" gains better sensitivity than "decreasing the weight by 1".

6.8 Seeding for Proteins - BLASTP

With nucleotides, we're requiring k positions with exact matches. However, for proteins, that's not really reasonable: some amino acids mutate to another one very often, namely we don't require exact matches, but also positive score matches. So BLASTP looks for 3- or 4-letter protein sequences that are "very close" to each other, and then builds matches from them. Where very close, then total BLOSUM score in the short window is at least $+13$ for 4-mer (or $+11$ for 3-mer).

One way is to find all neighbors of 3-mer, which are all 3-mers give BLOSUM score ≥ 11. Thus we can use computer to pre-compute all neighbors of every 3-mer, and put in the index table. The details:

1. For every 3-mer, find all "neighboring" 3-mers that, score at least +11 (or whatever). Build these into a data structure NeighborList.
2. Build a hash table H for S of its 3-mers, just like for the nucleotide case
3. For every 3-mer x in T, retrieve all neighbors from NeighborList. For each neighbor, query H to find hits in S.

NeighborList is a small structure: there are only 8000 3-mers

The original way is with BLASTP:

- Build an automaton that reflects all string close to short strings in T (the short sequence)
- Scan S (the longer sequence), looking for matches.

Details: https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm

Which sequence to index? That's actually a tough question. Here's a typical scenario:

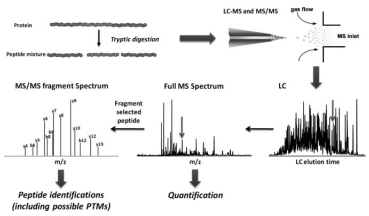
- S is the human genome (length n)
- P_1 is a short protein sequence (length m_1)

- P_2 is another short protein sequence (length m_2)

If we're smart, build an index for S, *once*, and then look up the short sequences in it. Added time for P_2 is more like $O(m_2)$, not $O(n + m_2)$.

Memory is a concern in old days, when indexing the human genome, but not anymore today. We probably should index the longer sequence. BLASTN (1990) indexes the query, not the database. BLAT (2000) indexes the database, not the query. BLASTP also indexes the query.

As an extension to this idea, consider two-hit BLAST: it requires two seeds (probably shorter) that are nearer than k from each other, and base the alignment on their enclosing box. In other words, two short hits are very close. Potentially even fewer false positives, but one has to use shorter seeds. There's quite a tradeoff here. This might lose sensitivity, but we can gain sensitivity back by reducing weight of seeds.



Part II:

Proteomics and Mass Spectrometry

Img src: https://www.jbc.org/article/S0021-9258(19)48534-7/fulltext

7

Introduction

蛋白质组学和质谱法

7.1 Motivation

Primary structure of **protein** is a sequence of amino acid. 20 frequent amino acids. Fold into a complex 3D structure.

Fundamental questions Identify, sequence, and quantify all the proteins in a biological sample.

- **Identification**: determine which proteins in a database present in the sample.
- **Sequencing**: determine the amino acid sequence without needing a database.
- **Quantification**: determine the quantity change of each protein under two different biological conditions.

Example: Biomarker

HER2-positive breast cancer is a breast cancer that tests positive for a protein called human epidermal growth factor receptor 2 (HER2), which promotes the growth of cancer cells.

From Mayo Clinic:

> HER2-positive breast cancers tend to be more aggressive than other types of breast cancer. They're also less responsive to hormone treatment. However, treatments that specifically target HER2 are very effective.

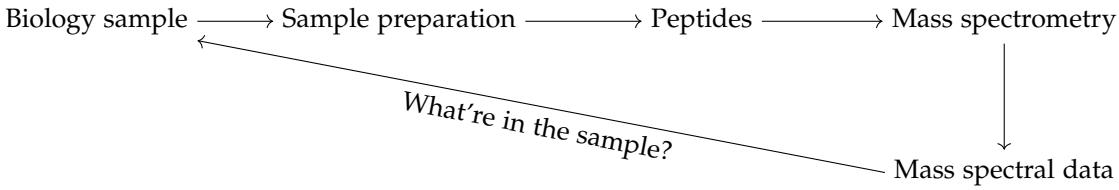
Example: Immunopeptides (免疫活性肽)

Tumor (肿瘤) or infected cells "present" some abnormal peptides at the MHC on cell surface. CD8+ T cells (aka T killer cells) **recognize** the abnormal peptides and kill the cell. An actively pursued method in immunotherapy is to identify/predict the peptides presented and train the T cells to target them. Mass spec is the best tool to identify these peptides.

Example: Antibody

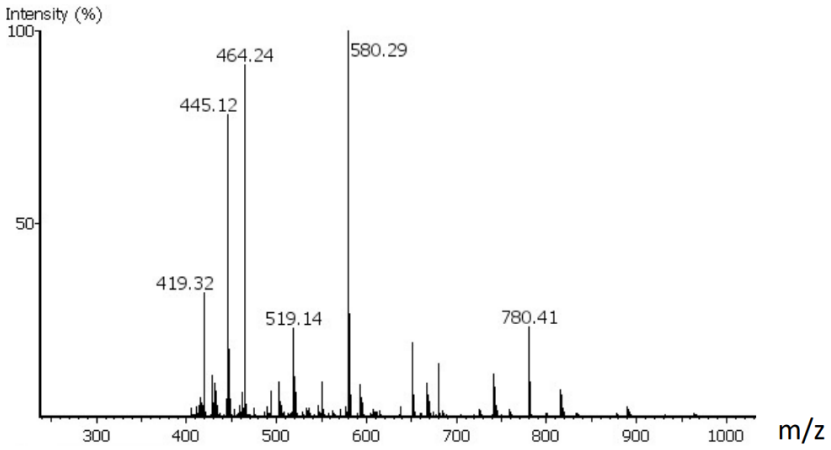
Our immune system produces antibodies to bind to invasive pathogens (virus etc.) and cancer neoantigen (抗原). Antibody's amino acid sequences determine the binding target (antigen). Some people have stronger immune system. E.g. HIV elite controller. Sequencing their antibodies provides a new way to discover antibody drugs.

Here is technology overview:



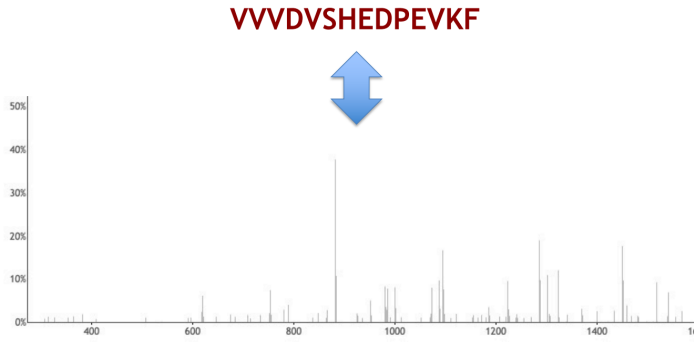
7.2 Mass spectrum of a peptide

Consider a mass spectrum.

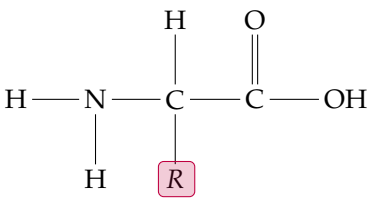


m/z is the x-axis, which is the ratio from mass (molecular weight) to charge. Each peak indicates the detection of a particular type of ion (electrically charged molecule) with the corresponding mass to charge ratio. Mass of an ion can also be derived if the charge state (z) of an ion is known, which is achievable. A typical proteomics experiment produces tens of thousands spectra per hour.

For each peptide, it tries to produce a **Tandem Mass Spectrum** (MS/MS). Top → Bottom spectrum is MS/MS. Bottom to top is bioinformatics.

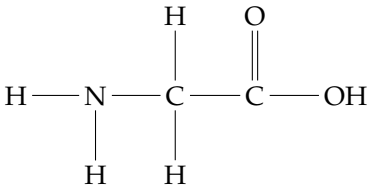


There are 20 amino acids. All have the same basic structure but with different side chains:

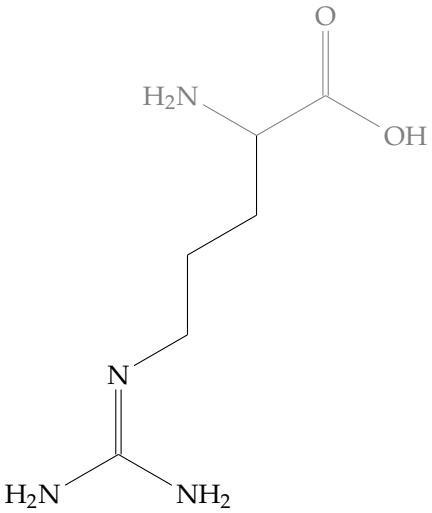


R Group (Side Chain)

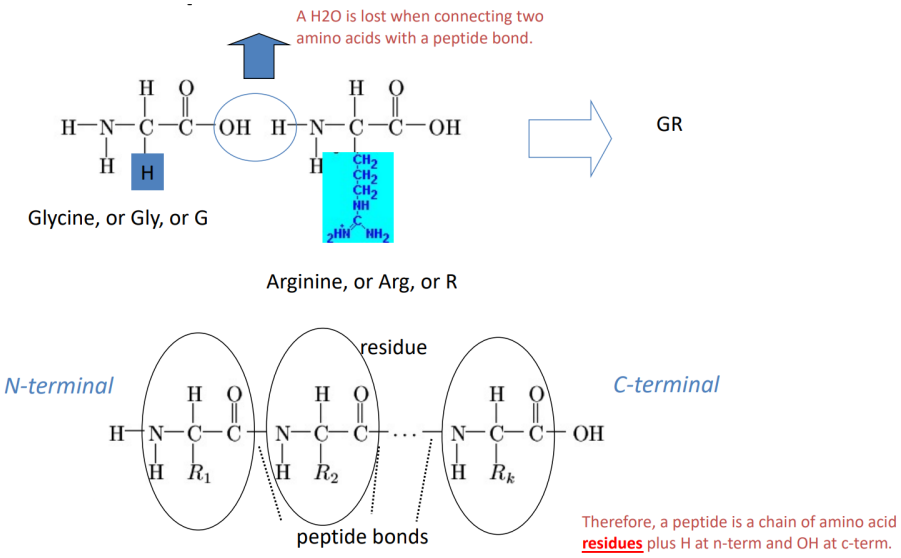
Consider two examples. Glycine, or Gly, or G



and Arginine, or Arg, or R



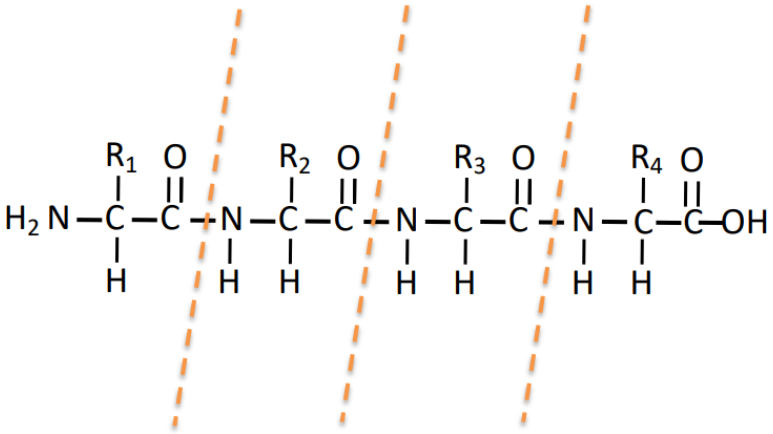
A H2O is lost when connecting two amino acids with a peptide bond. Therefore, a peptide is a chain of **amino acid residues** (氨基酸残基) plus H at N-term and OH at C-term. [1]



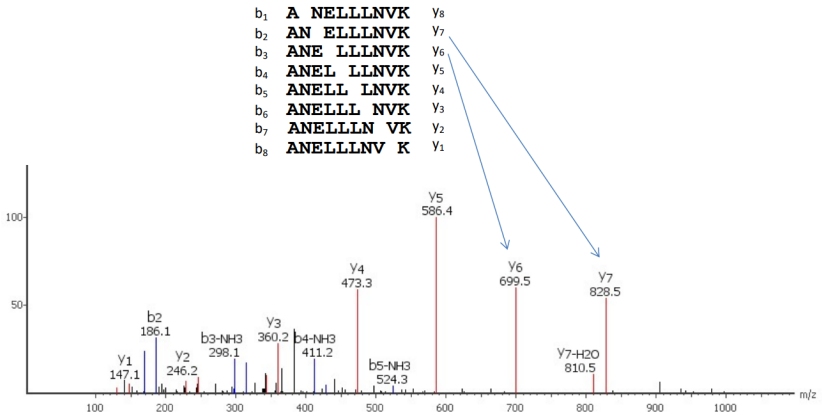
Once we have the chain of residues, we can compute the molecular weight, the mass of peptide. Consider Amino Acid **Residue** Mass Table. Mass unit is Da (or Dalton). For example, the approximate (integer) mass of peptide GGK is

$$\underbrace{57}_{G} + \underbrace{57}_{G} + \underbrace{128}_{K} + \underbrace{18}_{H_2O} = 260$$

[1] Sorry, `chemfig` is so hard to learn...

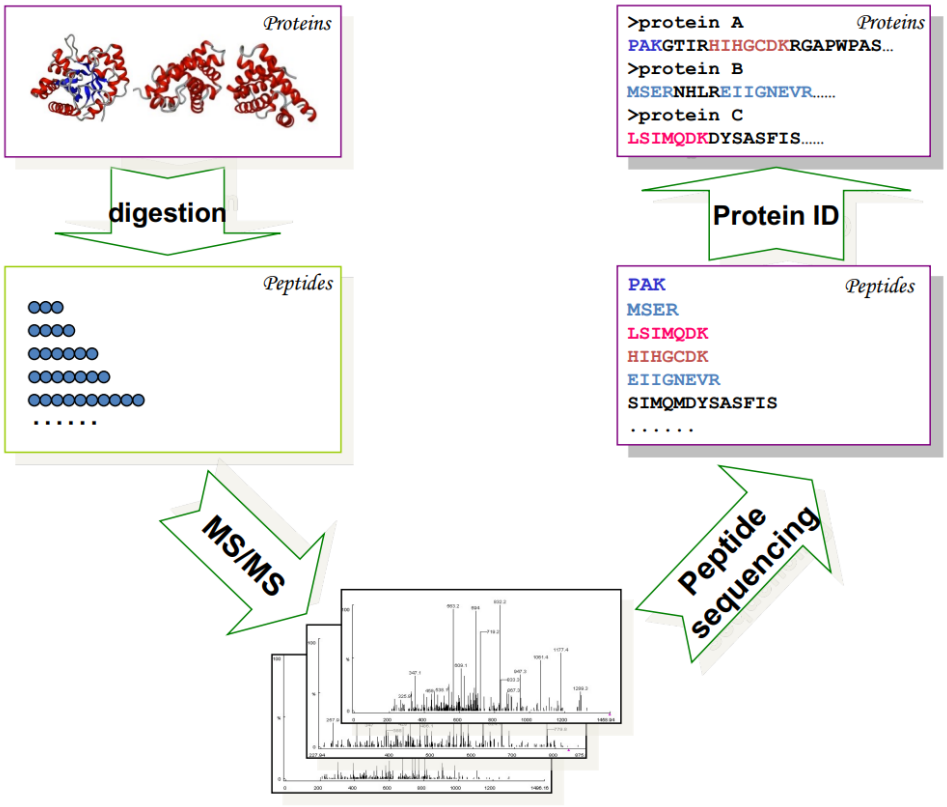


Peptide Fragmentation in MS/MS. Fragment between peptide bond. We don't want to do all fragmentation and get single amino acids, which don't reveal structural information. We want to fragment once. Then we get MS/MS of a peptide. Left is b-ion, right is y-ion.

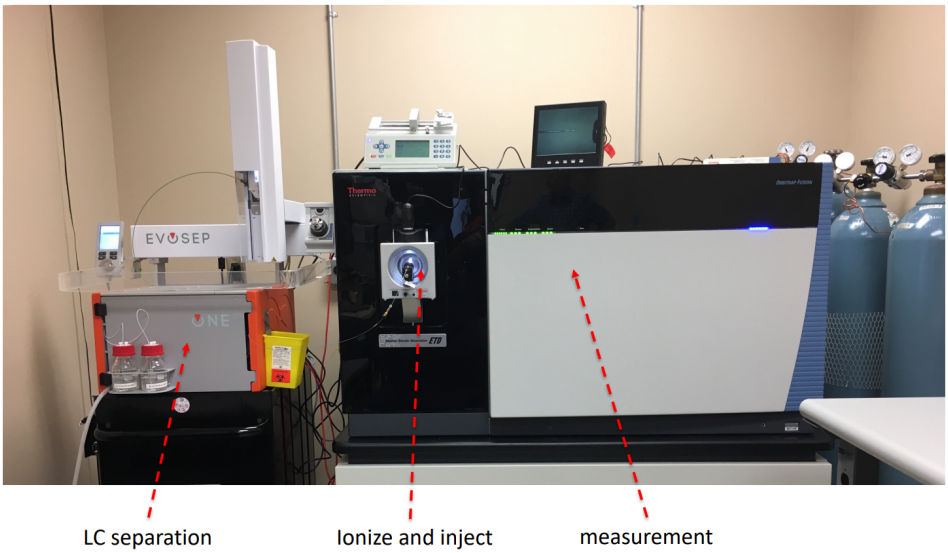


For example, we find the difference between y_7 and y_6 is 129, this tells us there is an E amino acid in between. Mass difference between two adjacent "ladder" ions can be used to determine the amino acid (residue).

Bottom Up Proteomics



A Typical LC-MS Setup



7.3 Three Basic Components of Spectrometer

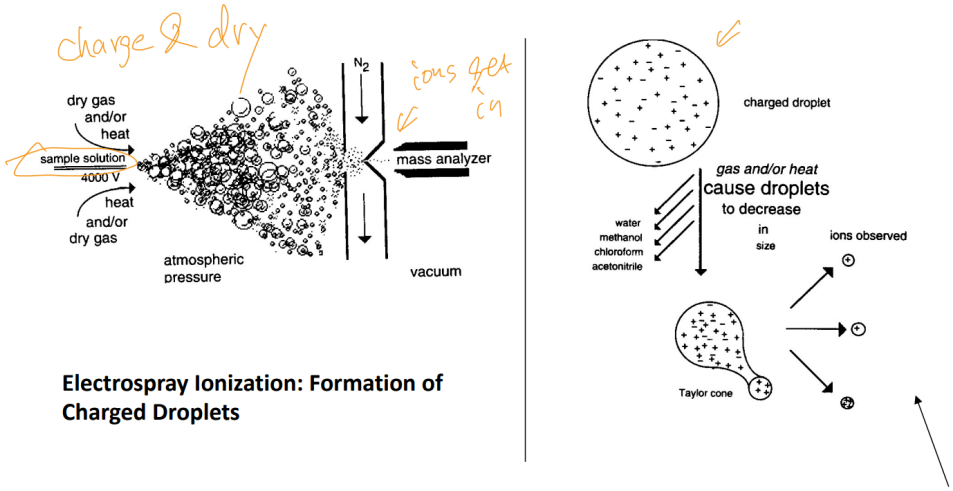
- Ionizer
  - provide electric charges to the molecules
  - MALDI, **ESI**

- Mass Analyzer
  - Separates the ions according to m/z
  - Magnetic Sector, Iontrap, **TOF**, **Quadrupole**, FT, **Orbitrap**
- Detector
  - Detect the separated ions
  - Electron multiplier, Fourier Transform

7.3.1 Ionization

MALDI (Matrix Assisted Laser Desorption/Ionization) by Koichi Tanaka, ESI (Electrospray Ionization) by John Fenn

Ionization - ESI

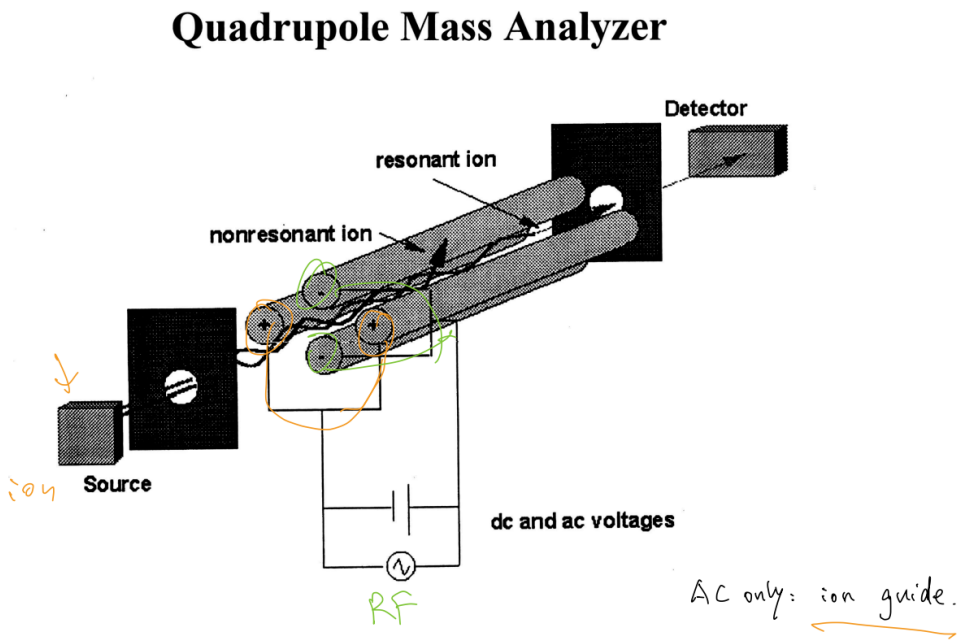


Spray sample solution, provide charge & dry condition. Then charged droplets. Water evaporate. At some point, charged molecules will be ejected out of droplet, which will become ion. Then ions gets in. We need to make this hole tiny, because in the spectrometer, there is a big vacuum.

There's a special property of ESI: ions may have different charge states. That used to be a problem, but now with proper data analysis, it is a good property.

7.3.2 Mass Analyzer (1) - Quadrupole

We need to control ions, otherwise it will diffuse to everywhere because of the vacuum. Quadrupole is to hold ions together.

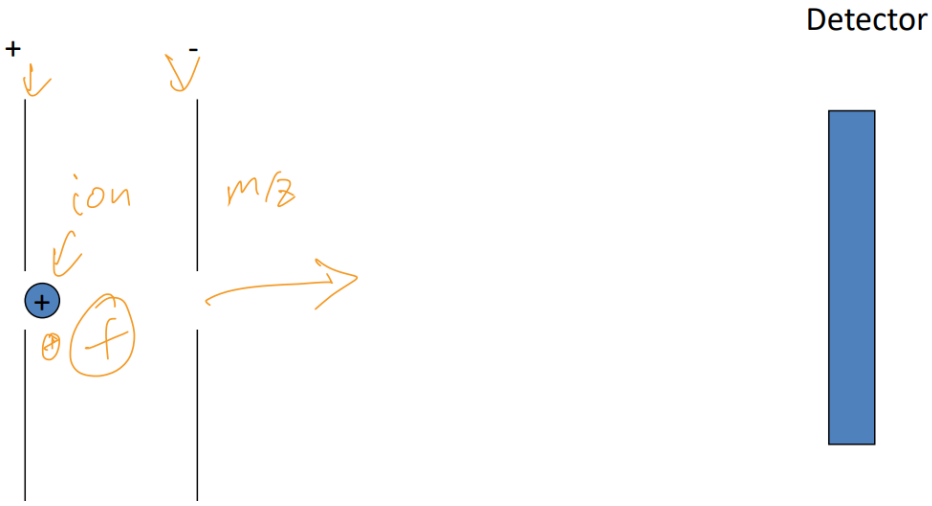


- Adjusting DC voltage allows different m/z ions to pass. (Mass filter)
- The complete spectrum is obtained by scanning whole range.

Mass range 10 - 4,000 Da

7.3.3 Mass Analyzer (2) – TOF

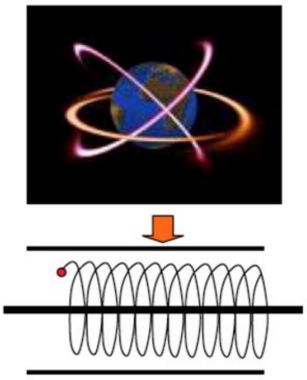
Time of flight.



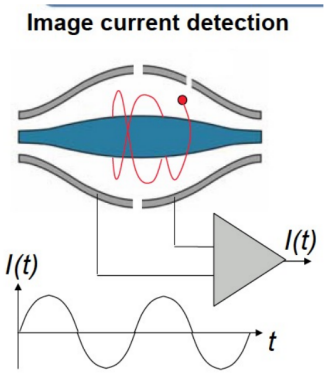
Time of flight $\sim \sqrt{m/z}$

7.3.4 Mass Analyzer (3) – Orbitrap

By far, most popular.



Moving ions are trapped around an electrode.



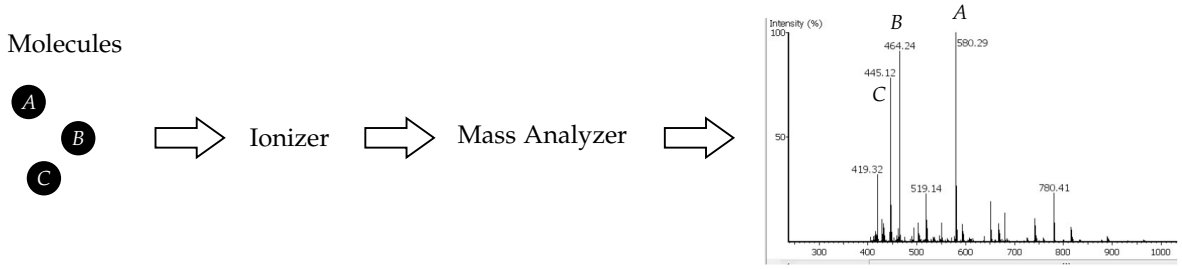
Notes:
-All-mass detection
-Noise equiv. to 20 ions (1 sec)

By shaping the electrode appropriately, ions also move left and right. Left-right frequency $\sim \sqrt{m/z}$

Fourier transform to convert the time domain signal to frequencies.

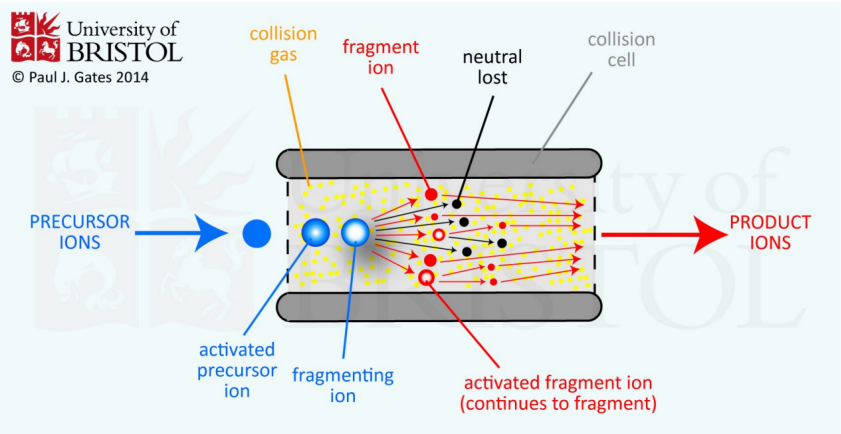
We can increase the measure accuracy by measuring longer, then the frequency will become more accurate, then we do Fourier transform. Thus it's the most accurate affordable machine.

Then the mass spectrometer:

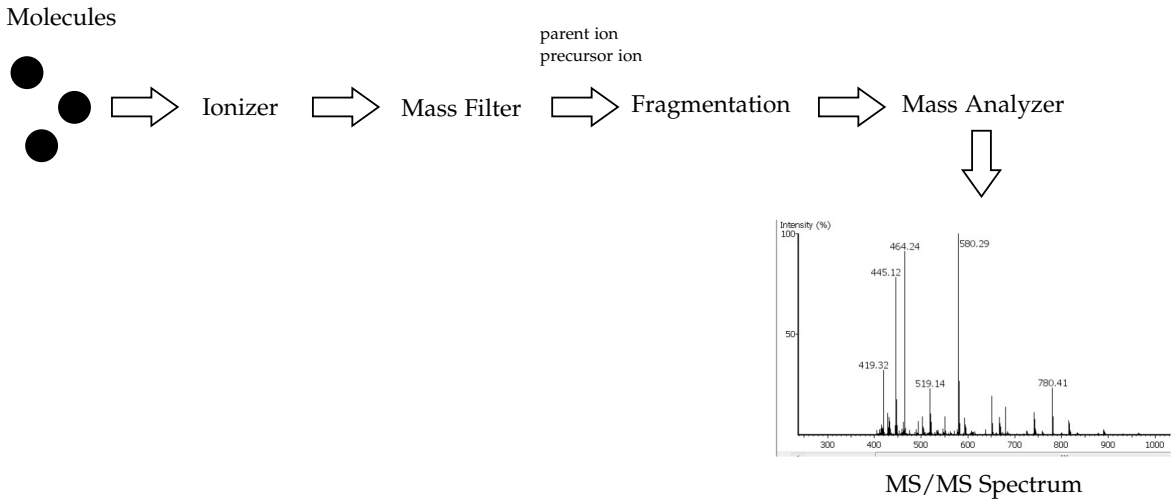


MS Spectrum

However, this is not good enough to identify the peptide because in the database, there are many peptides of identical molecular weight. To get that information, we need to fragment the peptides further. For example, let's select A, which we will use quadrupole. Now from ionizer, instead of shooting into mass analyzer, we shoot it into quadrupole. We program quadrupole so that we only allow certain m/z ions to go through.



Then these selected peptides are called **precursor ions**. We then put them into collision cell, and fragment them into pieces. After the process, the remaining fragment ions are called **product ions**, which are fragments of precursor peptides. And they can be further measured with another mass spectrometer. Putting these together, we have tandem mass spectrometer.

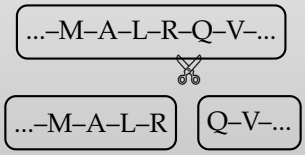


MS/MS Spectrum

To distinguish spectrum of precursors, and spectrum of fragment ions, people call this MS/MS spectrum, or tandem (串联) MS, or MS^2.

Protease

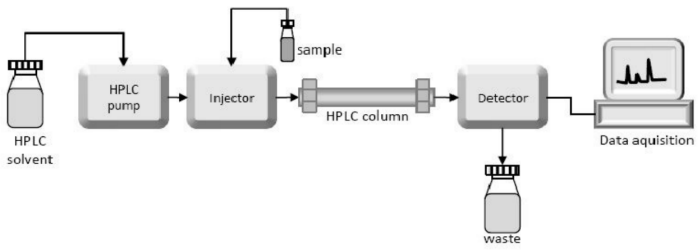
Proteins are generally too large for mass spec. They need to be cut into short peptides first. A protease is any enzyme (酶) that conducts proteolysis (蛋白水解). In another word, a protease breaks protein in water. Trypsin (胰蛋白酶) is the most commonly used enzyme. It digests at site `[KR] | [^P]`: After K or R, but not before P.



7.4 Liquid Chromotography (LC)

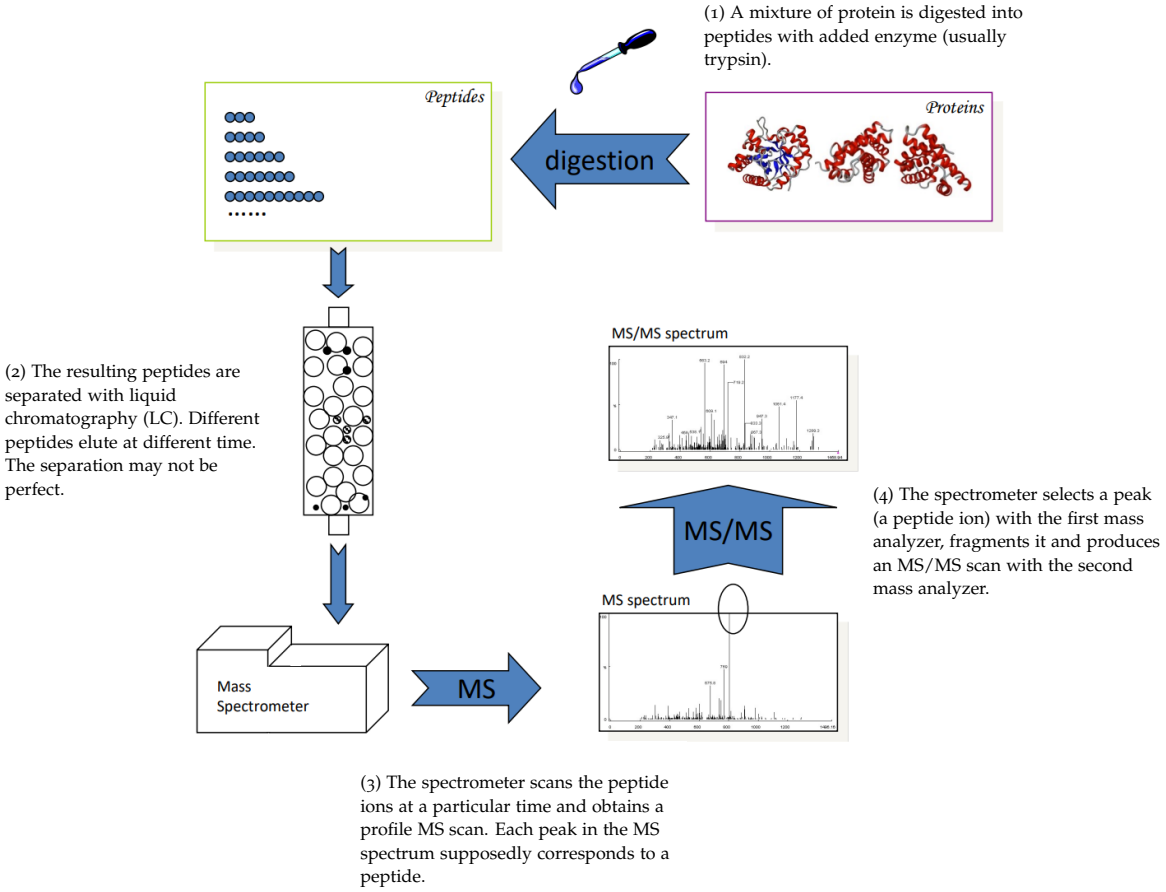
This creates another problem of **sample complexity**. Suppose there are 10,000 proteins to be analyzed. Each produces 100 peptides. Then there are 1 million peptides. If all of them are injected into mass spec simultaneously, we will see a peak everywhere. No useful information at all. To solve this, we separate them and inject a subset of them to mass spec at any given time. So spectrometer is not busy.

This is done by Liquid Chromotography (LC).



1. Peptides are loaded to the column.
2. Solvent is pumped into the column to wash peptides off.
3. By gradually adjusting the solvent's hydrophobicity (疏水性), peptides with different hydrophobicity will come off at different time. This is the retention time (RT)[2] of the peptide.
4. Peptides are therefore separated. (But the separation is not perfect..)

Put all pieces together, we have



MS spectrum is sometimes called survey scans: produce a survey of all peptides getting into the spectrometer. We "acquire" MS2 spectrum from survey scan. This flow is called **data-dependent acquisition** (DDA).

[2]In chromatography, retention time (RT) is the interval between the injection of a sample and the detection of substances in that sample.

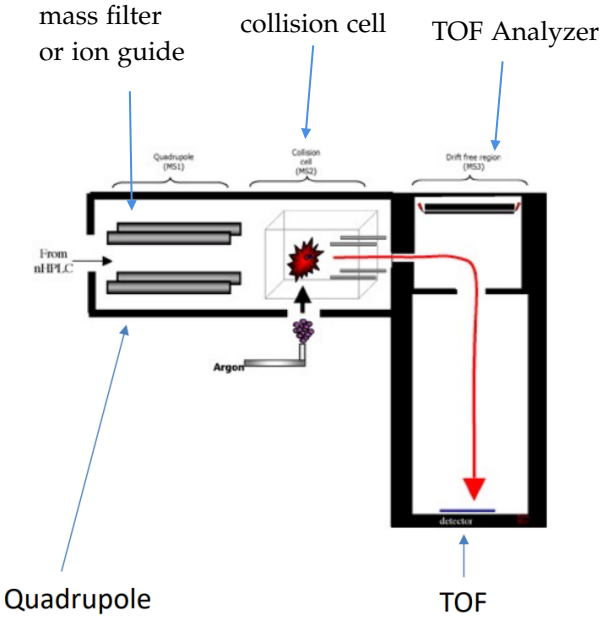
8

Database Search

In the previous chapter, we have learnt how the data is produced. Now let's learn how we can use and analyze it.

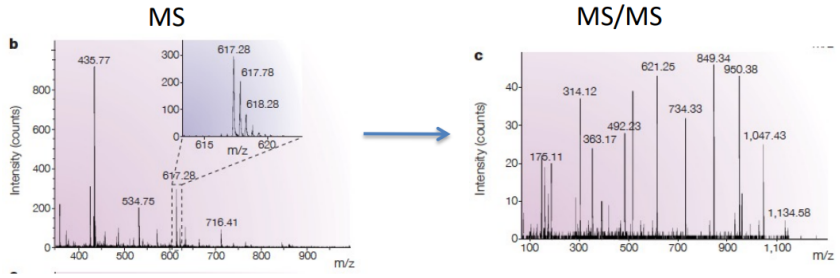
8.1 Reviews

Tandem Mass Spectrometry



- Tandem MS combines different mass analyzers. E.g. Q-Tof.
- Quadrupole can run in either ion guide or ion filter modes.
- To measure the precursor ions: Quadrupole in ion guide mode, Collision off
- To measure the fragment ions of a precursor ion: Quadrupole to select target m/z, Collision on

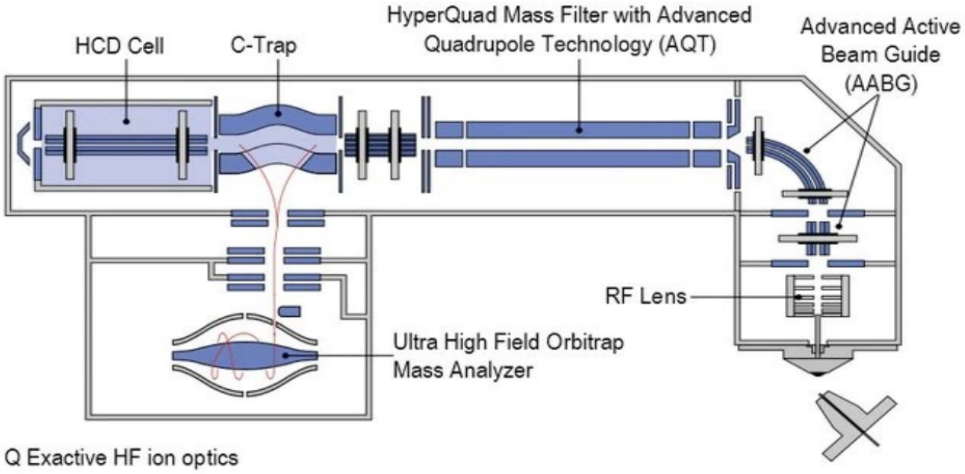
Tandem Mass Spectrometry Procedure



- Step 1. All precursor ions are measured to produce the survey scan.
- Step 2. A precursor ion is selected (by m/z) and fragmented. All fragment ions are measured to produce the tandem MS scan (also called as MS/MS or MS2 scan).
- Repeat Step 2 a few times. Then go back to Step 1.

Note: For each MS2 spectrum, we additionally know the precursor m/z.

Orbitrap QE-HF



An actual instrument may consist many components for better sensitivity, accuracy, throughput and robustness. The figure illustrate the main components of an Orbitrap QE-HF instrument made by ThermoFisher Scientific.

8.2 Peptide Spectrum Match

We have the formula:

$$\text{y-ion m/z} = (\text{total of amino acid residue mass} + \underset{\downarrow}{18.011}\ (H_2O) + z \times 1.007)/z$$

$$\text{b-ion m/z} = (\text{total of amino acid residue mass} + z \times \underset{\uparrow}{1.007}\ (\text{proton}))/\underset{\uparrow}{z}\ (\text{charge state})$$

Peak matching allows certain mass error tolerance (due to instrument measurement errors). Error can be specified either in Da or in ppm (part-per-million).

$$\text{ppm error} = 10^6 \times (\text{observed mass} - \text{theoretical mass})/\text{theoretical mass}$$

Different instrument has different error tolerance:

- Low resolution: often 0.5-1 Da
- High resolution: often 1-20 ppm

Precursor ions and fragment ions often have slightly different error tolerances, because instrument might have different set of parameters to do survey scan and MS2 scan, which is to maximize the utilization of instruments.

Now we search through database. For example, consider this protein:

```
>sp|P02769|ALBU_BOVIN Serum albumin OS=Bos taurus GN=ALB PE=1 SV=4
MKWVTFISLLLLFSSAYSRGVFRRDTHKSEIAHRFKDLGEEHFKGLVLIAFSQYLQQCPFDEH
VKLVNELTEFAKTCVADESHAGCEKSLHTLFGDELCKVASLRETYGDMADCCEKQEPERNEC
FLSHKDDSPDLPKLKPDPNTLCDEFKADEKKFWGKYLYEIARRHPYFYAPELLYYANKYNGVF
QECCQAEDKGACLLPKIETMREKVLASSARQRLRCASIQKFGERALKAWSVARLSQKFPKA
EFVEVTKLVTDLTKVHKECCHGDLLECADDRADLAKYICDNQDTISSKLKECCDKPLLEKSHC
IAEVEKDAIPENLPPLTADFAEDKDVCKNYQEAKDAFLGSFLYEYSRRHPEYAVSVLLRLAKEY
EATLEECCAKDDPHACYSTVFDKLKHLVDEPQNLIKQNCDQFEKLGEYGFQNALIVRYTRKV
PQVSTPTLVEVSRSLGKVGTRCCTKPESERMPCTEDYLSLILNRLCVLHEKTPVSEKVTKCCTE
SLVNRRPCFSALTPDETYVPKAFDEKLFTFHADICTLPDTEKQIKKQTALVELLKHKPKATEEQL
KTVMENFVAFVDKCCAADDKEACFAVEGPKLVVSTQTALA
```

- Use the enzyme digestion rule to cut each protein into peptides

```
MK|WVTFISLLLLFSSAYSR|GVFR|R|DTHKSEIAHR|FK|DLGEEHFK|GLVLIAFSQYLQQCPFDEHVK|L
```

- For each peptide, compare with the spectrum to see how well they match.

We need an empirical score. Recall that y-ion m/z at charge one = total residue mass + 19.0178. Then we find approximate matching peak. Let x denote relative intensity where

$$\text{relative intensity} = \frac{\text{current peak intensity}}{\text{max peak intensity}}$$

Then

$$\text{Score contribution} = \max \begin{cases} \log_{10} 100 \cdot x & \text{if } x > 0.01 \\ 0 & \text{otherwise} \end{cases}$$

We then add up all score contributions of all y-ions. Better score functions will be discussed later.

Note that we can use y-ion with charge state 1, or 2 as well. More ion types, more charge states we use, we will capture more signals. At the same time, it will increase the change that random peptide will give us a high score as well.

This then brings us the database search algorithm:

Algorithm 10: Database Search

Input: A list of MS/MS spectra, a protein sequence database

```
1 foreach MS/MS spectrum do
2     foreach protein in database do
3         In-silico digest the protein into peptides
4         foreach peptide do
5             Evaluate the peptide-spectrum match
6     Assign the highest-scoring peptide to the spectrum
```

In practice, we can speed up by a constant factor, though it might be huge, which will be achieved via filtration.

Algorithm 11: Database Search - with filtration

Input: A list of MS/MS spectra, a protein sequence database

```
1 foreach MS/MS spectrum do
2     foreach protein in database do
3         In-silico digest the protein into peptides
4         foreach peptide do
5             if precursor mass error < allowed error tolerance then
6                 Evaluate the peptide-spectrum match
7     Assign the highest-scoring peptide to the spectrum
```

Here precursor mass error $=$ | theoretical precursor mass $-$ observed precursor mass |. The mass filtration reduces the number of PSM evaluation.

We can speed up even further by doing binary search.

Algorithm 12: Database Search - with filtration & binary search

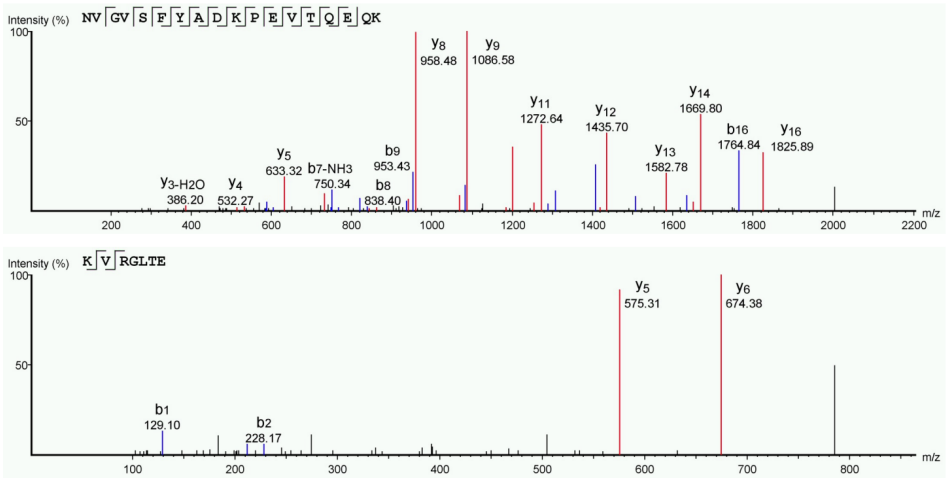
Input: A list of MS/MS spectra, a protein sequence database

```
1 Sort the spectra according to precursor mass // not m/z because we have different
    charge states
2 foreach protein in database do
3     In-silico digest the protein into peptides
4     foreach peptide do
5         foreach spectrum with matching precursor mass do
6             Evaluate the peptide-spectrum match
7             Keep the highest scoring peptide for the spectrum
```

The process unfolds as follows. We first sort the spectra, which is an array. Then we search for the location of the array where we would insert this mass value into this sorted array. This location would most likely give us smaller mass error. Then we search for both directions (left and right) in this position to find all matches.

8.3 Result Validation

However, the input will sometimes be junk or garbage. Some spectra are of lower quality: peptides do not fragment well, peptides fragment too much, peptides do not get charged. peptides are of low concentration, etc. Moreover, some spectra's true peptides are not in database. Therefore, search results of some spectra are junk. Computer scientists may think these are not their problems. After all, they've reported the "optimal" peptide for each spectrum.



These two PSMs are all the best match from database for two different spectra. Their confidences are clearly different. The first one is a beautiful match, where we can find peak for every single ion. The bottom one is not a good match, because such a match can happen randomly with only two ions. However, even both are optimal, bottom one has lower score, which might give us some idea.

Here is a conversation: Biologists vs. Computer Scientists:

COMPUTER SCIENTIST: I found the optimal solution!

BIOLOGIST: Is it REAL?

COMPUTER SCIENTIST: No. Half of them are wrong.

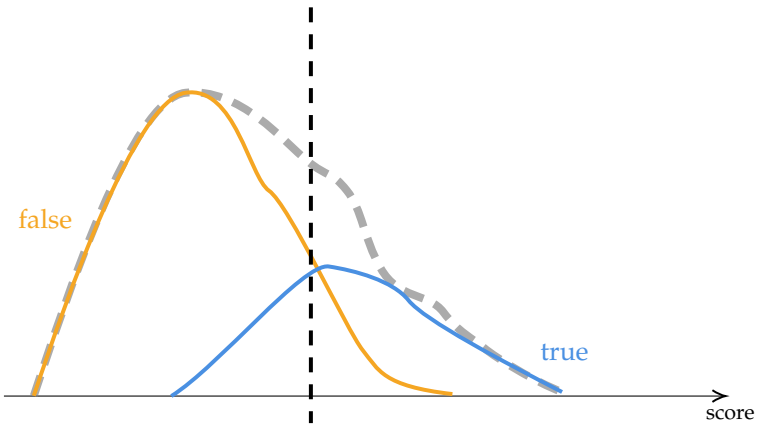
BIOLOGIST: The result is just like flipping a coin...

BIOLOGIST: Which half?

COMPUTER SCIENTIST: I don't know...

Solution to noisy input: only report results if you're confident, discard the less confident ones. This increases accuracy to make the results useful at the price of discarding some data.

Consider the score distribution:



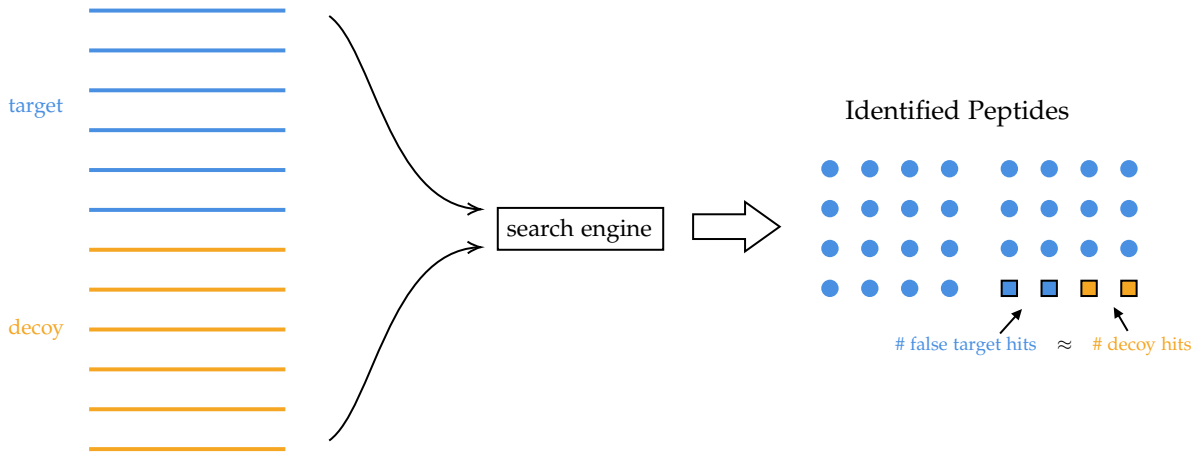
We can only see grey curve, which is the aggregated distribution. It is the sum of orange curve (false) and blue curve (true). We have a threshold that we only report everything to the right (of black vertical line). Thus blue region and orange region will be reported. Then the **false-discovery rate**

$$FDR = \frac{\text{\# reported false hits}}{\text{\# reported hits}}$$

By choosing different score threshold, one can calculate the FDR for all target PSMs above the threshold. Or conversely, one can choose a proper threshold to meet a FDR requirement. As of today, a typical FDR requirement is 1%. If we know blue curve and orange curve, we can adjust the threshold to achieve 1%. Unfortunately, we only know the aggregated distribution (grey curve).

We can remove all true matches/peptides, and the rest will be random matches, then the distribution will approximate the false curves. However, people don't know how to remove true peptides. Instead, we can create a false database. The the resulting curve will be similar to the orange curve.

This idea is called **FDR Estimation with Target-Decoy**, which has been refined many times. Given blue lines/proteins are from target database. For each of them, we generate a decoy sequence by shuffling within the protein or other ways.

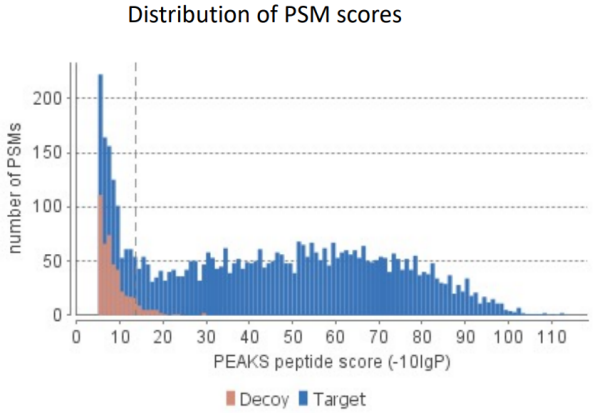


We *assume* that false matches randomly, where false matches are because some random peptides have high score matching this spectrum. If we identify a peptide that does not match the spectrum beautifully, then it must be by chance. Then false matches happen by chance will have equal probabilities in the target and decoy, because they all look similar, then false matches happen randomly in both cases. In the identified peptides, the round dots are correct ones, which are from target. The squared dots are false matches, from either target or decoy with the same probabilities. Then

$$FDR \approx \frac{\#\text{ decoy hits}}{\#\text{ targets}}$$

Denominator can be total hits, true target hits, or all target hits. This depends on the context or definition.

The paper is *Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry* (Joshua E Elias & Steven P Gygi, 2007)



In assignment 2, we mix N true and N random peptides together. By using your score, from the N top-scoring peptides, there are about $0.61N$ true peptides and $0.39N$ random peptides. How many of the "true" peptides are selected because your scoring function is truly amazing, and how many are pure luck?

Suppose we have 200 peptides, 100 target + 100 decoy. The top 100 consists of 61 targets and 39 decoys. Then out of 61 targets, there are 39 pure luck, and 22 true effort.

Now we can apply target-decoy method to the following problem. Suppose I'm asked to make and mark a final exam as a substitute teacher. But I know absolutely nothing about the subject. Fortunately, I have previous year's exam and all questions are multiple choices. But I do not know the correct answers. The challenge is that the student will use a randomized algorithm for this exam. Then how can we mark?

For each question, suppose each has A, B, C, D and A is correct. Then we make up three wrong options E, F, G, which are decoys, and shuffle options. If the student knows the true answer, then he will choose from A, B, C, D. If not, he will choose randomly, with same probability from B, C, D, E, F, G. Then $\#(A, B, C, D) - \#(E, F, G)$ is the true effort, not pure luck.

8.4 Better score function

The empirical score is only good for start up. Soon competition will get fierce and we'll need a better scoring function.

1. Separate true and false as much as possible, which is good for setting up the threshold.
2. Tell which spectrum is better.

Recall the idea of likelihood ratio. Let m be the m/z of a y-ion, and indeed, we see a peak with m/z = m in the spectrum. We have two assumptions (models):

- The peptide is the real peptide so peak is caused by the y-ion.

$$\Pr(\text{observe a peak at } m \mid m \text{ is a y-ion m/z of the real peptide})$$

- The peptide is a random peptide so the match is purely by chance.

$$\Pr(\text{observe a peak at } m \mid m \text{ is a random mass})$$

To calculate log likelihood ratio:

- Learn two probabilities from large training data. p: Prob(a peak is observed at a y-ion m/z), q: Prob(a peak is observed at a random m/z). Usually $p > q$.
- Given a peptide sequence, calculate m/z of all possible y-ions. For each y-ion,
  - If a peak observed, $\log \frac{p}{q}$ is added to score.
  - If no peak is observed, $\log \frac{1-p}{1-q}$ is added to score.
- Thus, matching ion is rewarded and missing ion is penalized.

Other fragment ion types can be considered similarly, and added to the score. For example, we can do b-ions. Usually, y-ions are stronger than b-ions (the difference between p and q is larger), thus we will reward y-ion matchings more than b-ions.

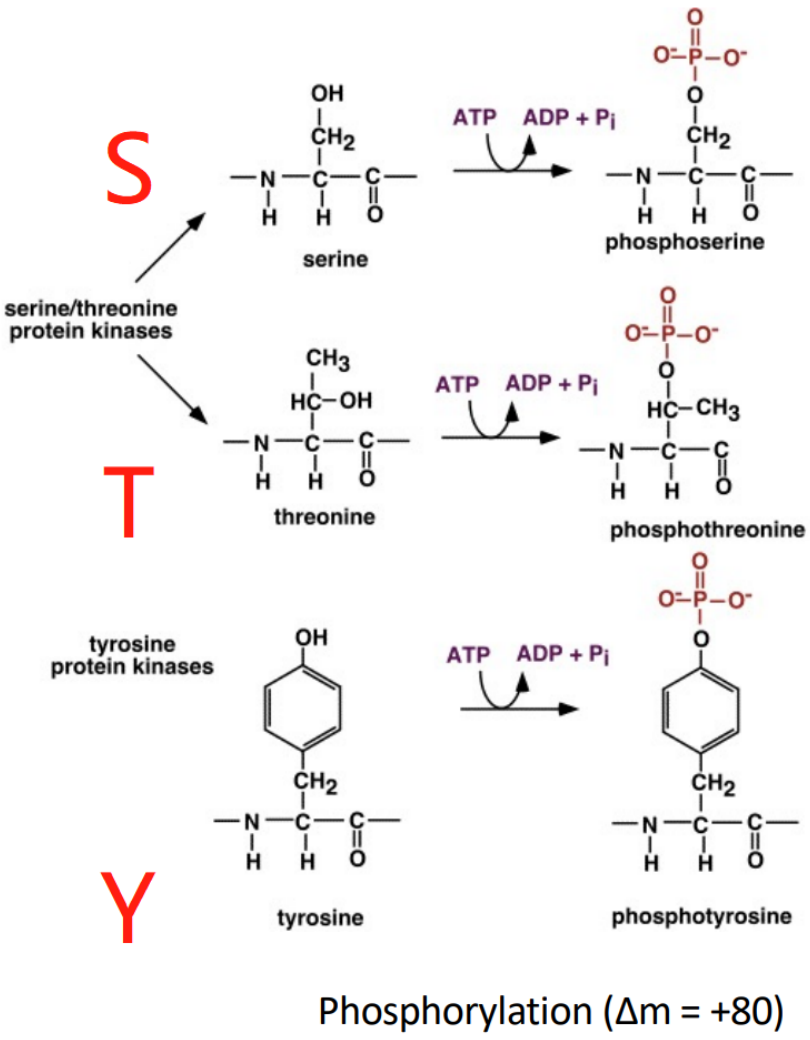
Also there are some other ideas to improve the score functions. Machine learning that combines many factors:

- Log likelihood ratio score
- Empirical score (log of relative intensity)
- Precursor error tolerance: for example, ppm error with respect to the precursor.
- Number of matching peaks
- Number of unmatched peaks
- Number of unmatched y-ions
- Include b-ions.
- Include charge 2 fragment ions.
- Etc.

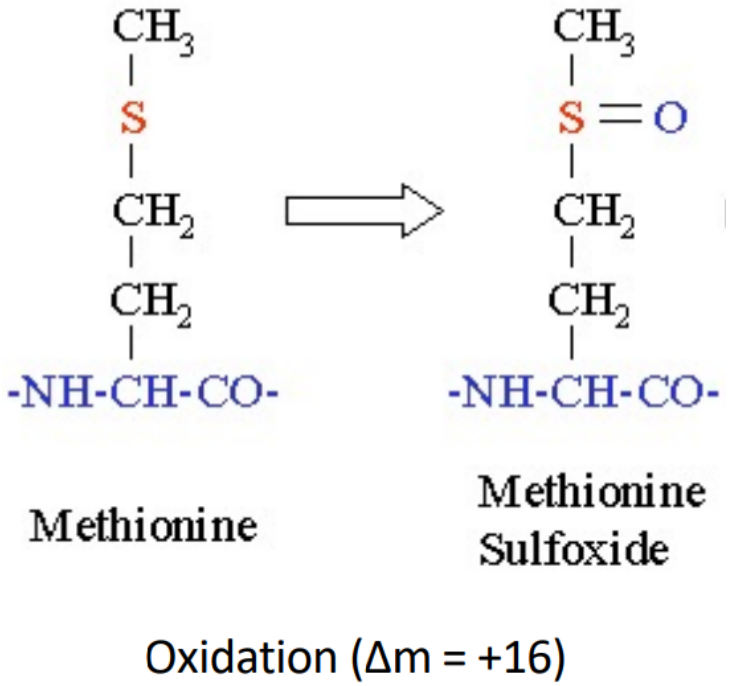
Let's consider some practical concerns.

8.5 Post-Translational Modifications (PTM)

Up to now, we assume there are only 20 animo acids. But this is not the case. There are 20 animo acids that are synthesized at the very beginning of the protein synthesization. After that, the cell will apply Post-translational modification (翻译后修饰) on proteins. For example,



磷酸化 (Phosphorylation), 氧化 (Oxidation)



Oxidation (Δm = +16)

PTM important to protein functions. Phosphorylation is used for biology to turn on or off the function of protein. Hundreds of different types of PTMs. PTM normally change the mass of an amino acid. Some PTMs can be on and off. The figures above show two common types of PTMs.

There are many hundreds of different types of PTMs included in the unimod PTM database. 30% of human proteins are phosphorylated, 50% are glycosylated (糖基化). PTMs are important to the functions of proteins. – For example: Reversible phosphorylation of proteins is an important regulatory mechanism. Many enzymes are switched "on" or "off" by phosphorylation and dephosphorylation(去磷酸化). The structural change caused by the PTM changes the function of the protein.

Now consider the idea of **variable PTMs**. If user selects some PTMs as "variable", all possible modification forms of a database peptide need to be tried to match the spectra. This results in exponential growth of search space. For example, denote Oxidation by `M` and Phosphorylation by `T`. Then given a variable PTM `PEPTIDEPTM`, we have the following possibilities:

```
PEPTIDEPTM
PEPT(+80)IDEPTM
PEPTIDEPT(+80)M
PEPT(+80)IDEPT(+80)M
PEPTIDEPTM(+16)
PEPT(+80)IDEPTM(+16)
PEPTIDEPT(+80)M(+16)
PEPT(+80)IDEPT(+80)M(+16)
```

Consequently, one can only search with a few variable PTMs.

There's a special type of PTM: **fixed PTMs**: certain modifications are deliberately added during the sample preparation and is (almost) 100%.

The most common one is that cysteines (半胱氨酸) are usually modified chemically. And the most common modification changes the mass from 103.00919 to 160.03065. Roughly 57.02 Da were added.

For curiosity only, cysteines are modified to avoid the formation of "disulphide (二硫键) bonds".

Fixed modification changes the amino acid residue mass table, but does not affect the database search speed.

There are some experimental errors, missed and nonspecific cleavages. The proteolyses may not be 100% efficient. Assume Trypsin digests the following protein with 100% efficiency.

```
SSAYSR/GVFR/R/DTHK/SEIAHR/F
```

Missed cleavages: a digestion site not cut. For example, `GCFRR` should have been cut between `R|R`. There are also **Non-specific cleavages**: a non-digestion site got cut. For example, `SEIAH` is not a Trypsin cut. Might because Trypsin is not pure.

Allowing them will both affect the algorithm's time complexity. If we allow algorithm to consider both errors, search space will be expanded. Which one has a bigger impact? Suppose N is length, and n is cutting sites. Missed cleavages is $O(n^2)$ as we need to explore places between every cut and non-specific cleavages is $O(N^2)$.

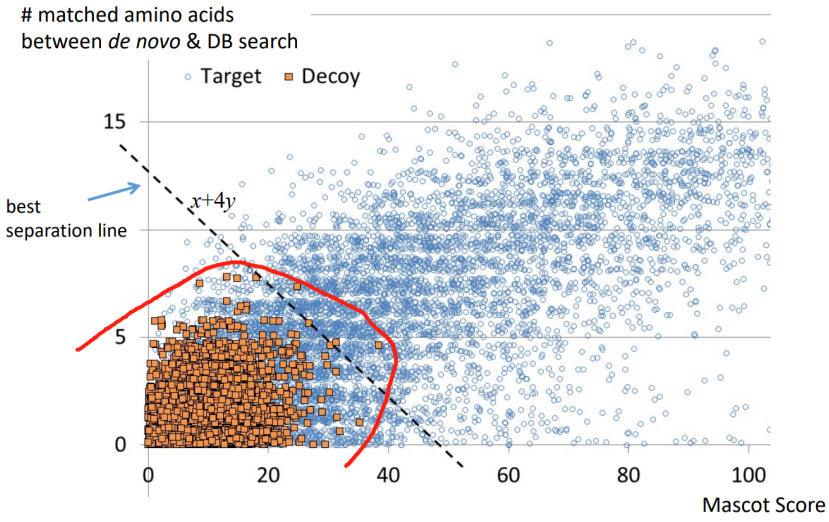
9

Database Search Details

9.1 Even Better Scoring Function

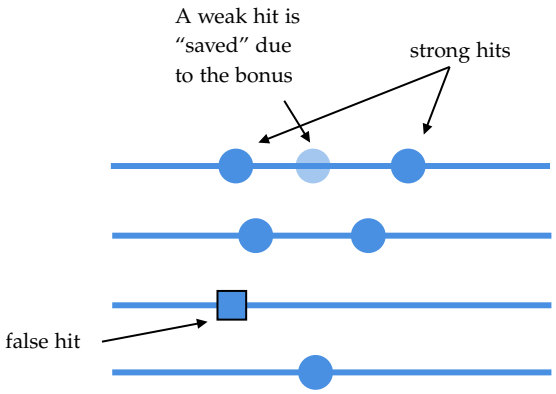
Recall that we can incorporate many other "features" for the scoring by a machine learning method. Features can apply to compute the matching/mismatching of certain fragment ion: Matched fragment ion intensities, mass error, correlation between intensity and surrounding amino acids. Features can apply to the whole peptide score: Precursor ion mass error, **Agreement with de novo sequencing Protein information**.

For agreement with de novo sequencing (sequence from the spectrum), we consider the graph below: two different approaches give the same answer.

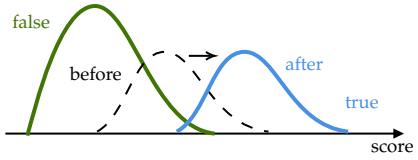


We see that the best separation line is the combination of both: $x + 4y$, or the red line which is the non-linear model.

Use Protein Information. We are trying to identify the peptide-spectrum match. Peptides on a multi-hit protein get a **bonus** on their scores to increase sensitivity.



Then more features make the score function better separate true and false matches.



9.2 Speed Concern

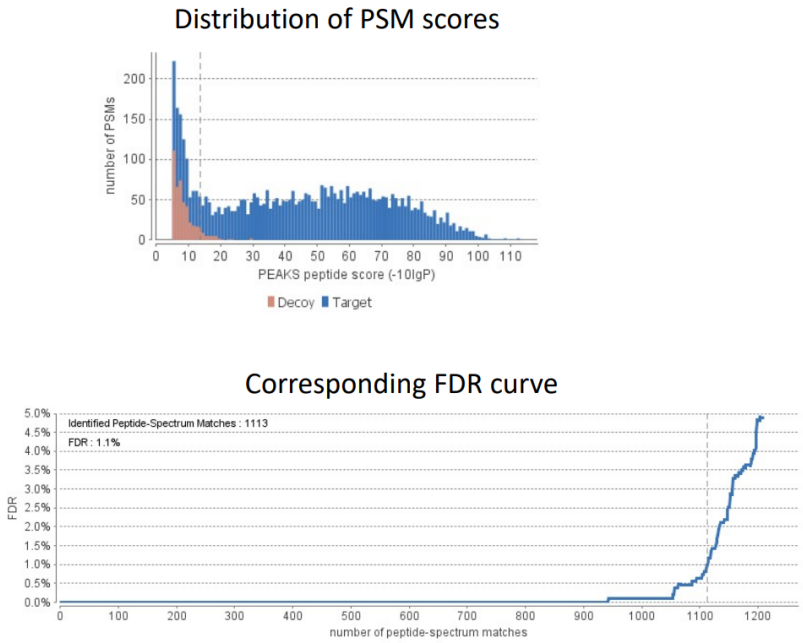
General programming wisdom:

- "Make it right, before make it fast." (??)
- "Premature optimization is the root of all evil" (Knuth)

Consider the idea of two-round search (Craig and Beavis 2004. *Bioinformatics* 20, 1466-67). For the **first round**, we do fast search. For example, we do not allow variable PTS in the first round, digesting errors and so on, or use high quality spectra. This will produce a short list of proteins. In the second round, we do **more sensitive search**, where we allow all things like non-specific cleavages, slower but better scoring function. Note that we really lose are the weak proteins in the first round.

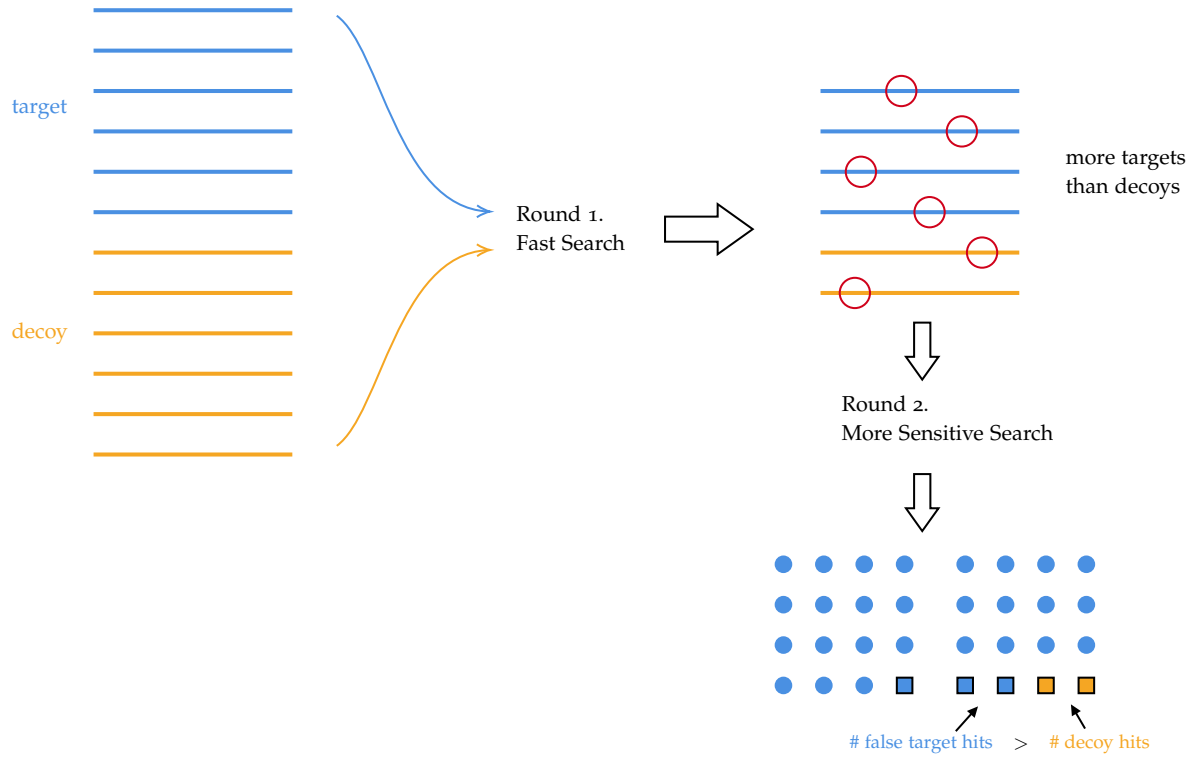
9.3 Pitfalls in FDR Estimation

These two ideas: protein score and the two-round search were developed before FDR was proposed. So they are not compatible with FDR estimation. Recall this is how FDR is estimated:



Pitfall 1 - Multiple Round Search

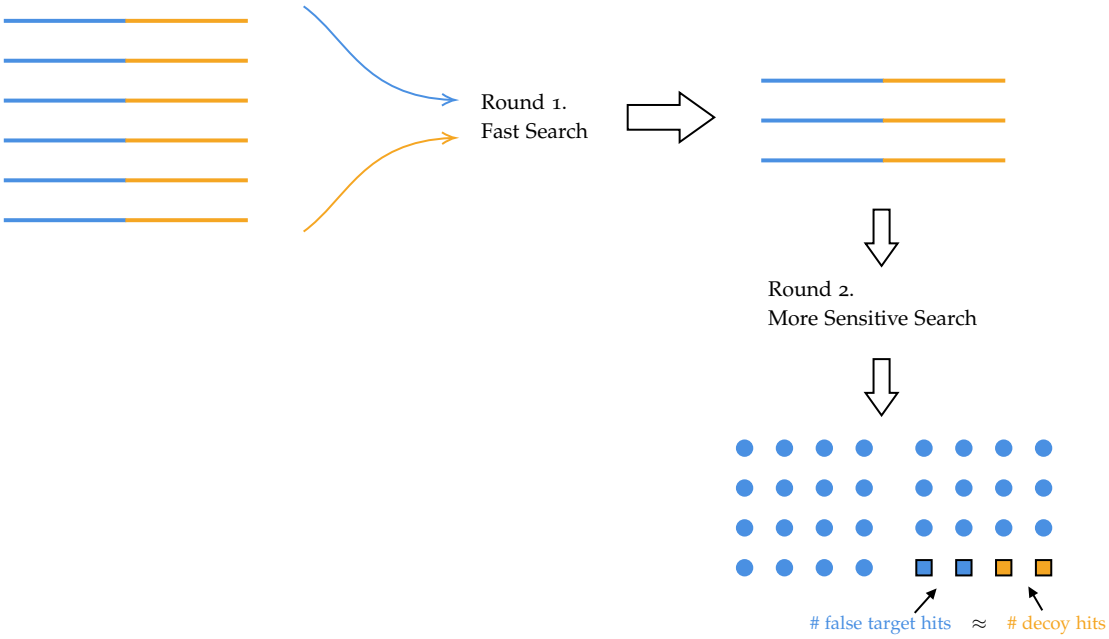
Red circles indicate false certifications matched by random chance. Then it's not true that # decoy matches ≈ # target false matches. This is not the fault of two-round search, just FDR estimation method is not powerful enough.



Two-round was first proposed by Craig and Beavis (2004). Then Evertt et al. (2010): fix two-round search by first throwing decoys away of the result of round 1, then appending decoys to create a new list. Bern and Kil (2011) said that this approach will make things worse: because this would make the

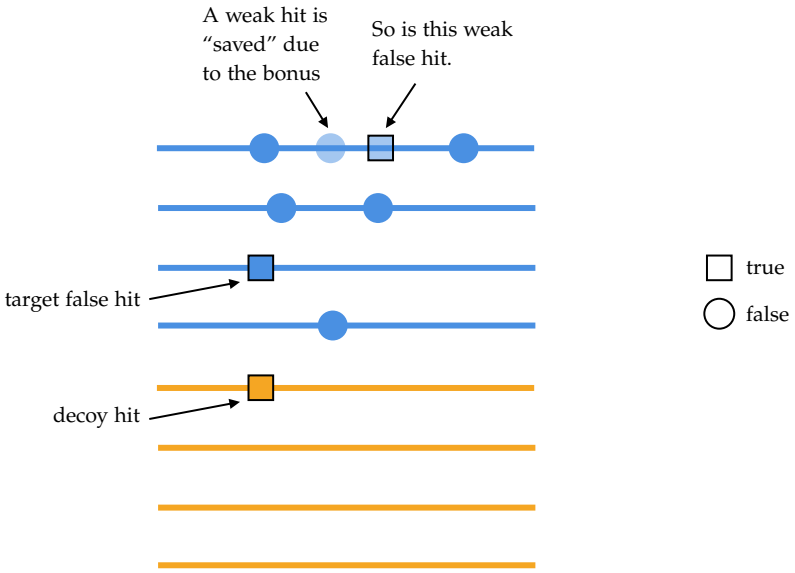
results biased. And they proposed to keep decoys of the result of round 1, and create a long list of decoys and mix them in. Then it will have more decoys than targets. It's better to be pessimistic than optimistic.

Later on, Zhang, J., Xin, L., Shan, B., Chen, W., Xie, M., Yuen, D., Zhang, W., Zhang, Z., Lajoie, G. A., & **Ma, B**. (2012). PEAKS DB: *de novo sequencing assisted database search for sensitive and accurate peptide identification* proposed an idea: **Decoy Fusion**. Then there's no bias between target and decoy, except the bias created by the spectrum data.



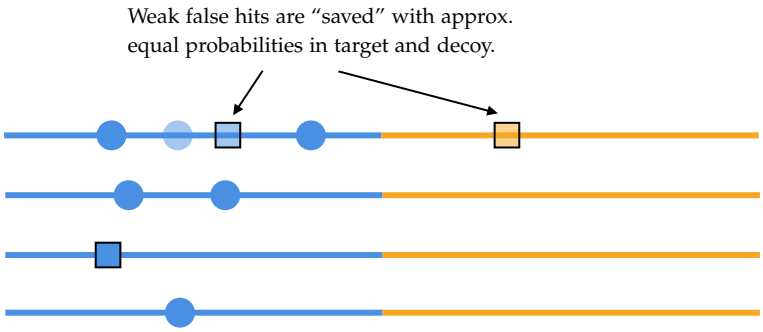
Pitfall 2 - Mix Protein and Peptide ID

It's hard to have multiple hits on decoy protein. Recall the previous idea: peptides on a multi-hit protein get a *bonus* on their scores to increase sensitivity. Then for all false peptides, false hits in the decoy proteins will never have bonus because of its low score. This creates imbalance between target and decoy false matches. Then More multi-hit proteins from target database, then more false hits are "saved" from target database, resulting in FDR underestimation.



Then decoy fusion can solve this, where target false hits and decoy false hits are the same. We got the

sensitivity, but still estimate the FDR correctly.



Pitfall 3 - Machine Learning with Decoy

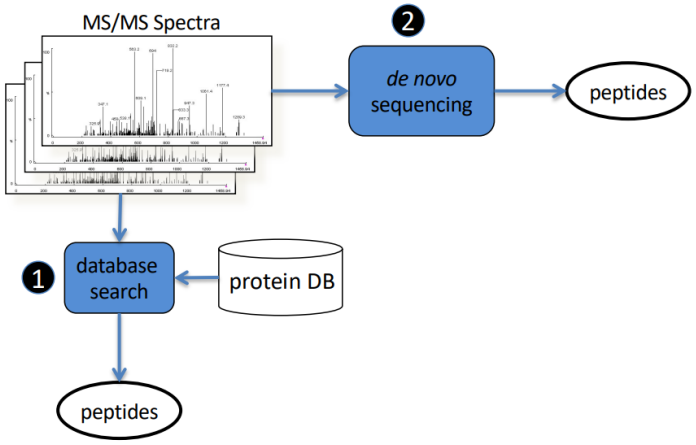
Re-train the coefficients of scoring function for every search after knowing the decoy hits. We adjust scoring function to remove decoy hits after search. The pitfall is the risk of over-fitting. Fewer target false hits are removed, resulting in FDR underestimation. Machine learning experts only.

So the solution is

1. Don’t use it. As judges cannot be players.
2. Only use for very large dataset. OR
3. Train coefficients and reuse; don’t re-train for every search.

10

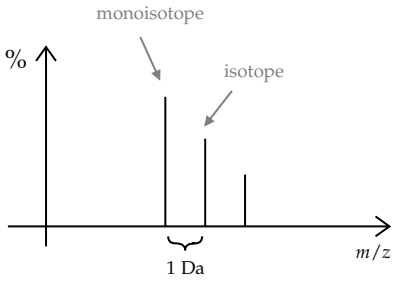
De Novo Peptide Sequencing



De Novo Peptide Sequencing is anther approach to identify the peptides. From wiki:

> In general usage, de novo (literally 'of new') is Latin expression used in English to mean 'from the beginning', 'anew'.

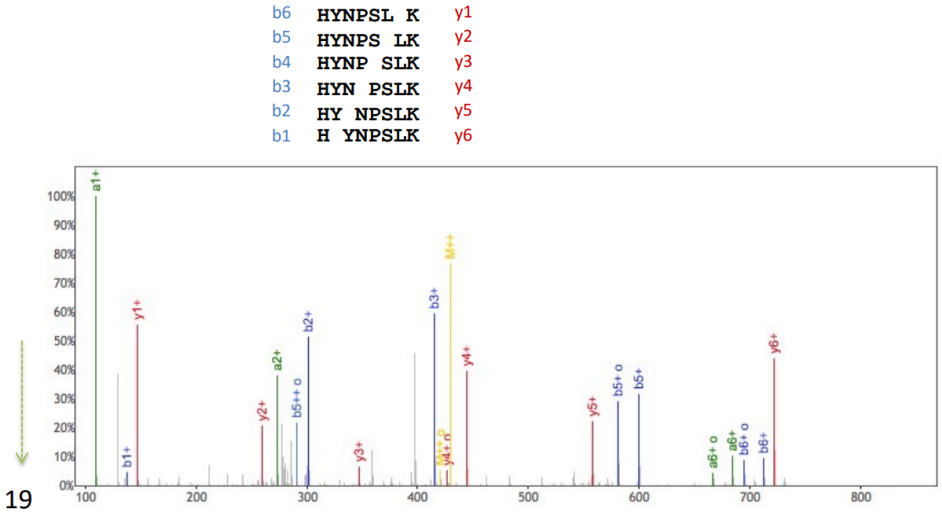
Isotope (同位素). For example, C: $\sim$ 99% ^{12}C (monoisotope), of 12.000 Da; $\sim$ 1% ^{13}C of 13.$\cdots$ Da. Then in the spectrum, it might look like this



The problem looks cleaner: we only have one spectrum and we want to determine the sequence. To construct a sequence that matches the spectrum the best.

10.1 Manual De Novo Peptide Sequencing

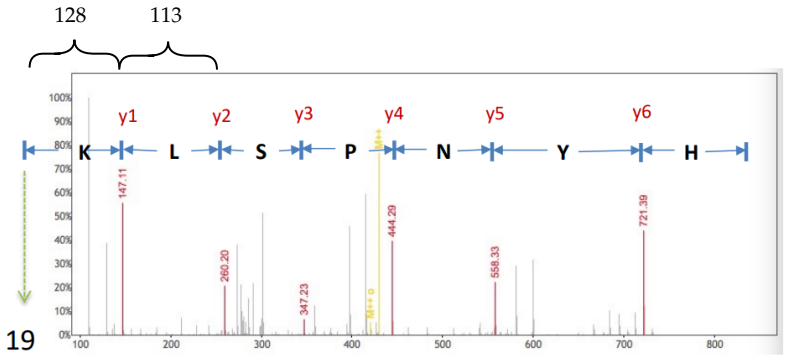
In early days, people do it manually. For example, consider the MS2 spectrum of `HYNPSLK`.



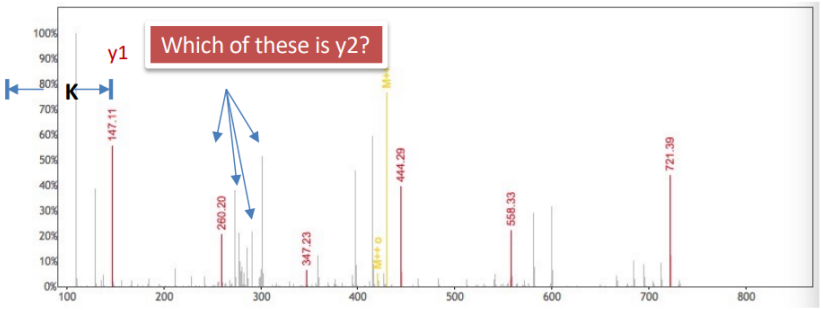
Now we focus on y-ions. Recall that

$$\text{y-ion m/z} = (\text{total of amino acid residue mass} + 18.011 + z \times 1.007)/z$$

We use nominal (integer) mass for simplicity.

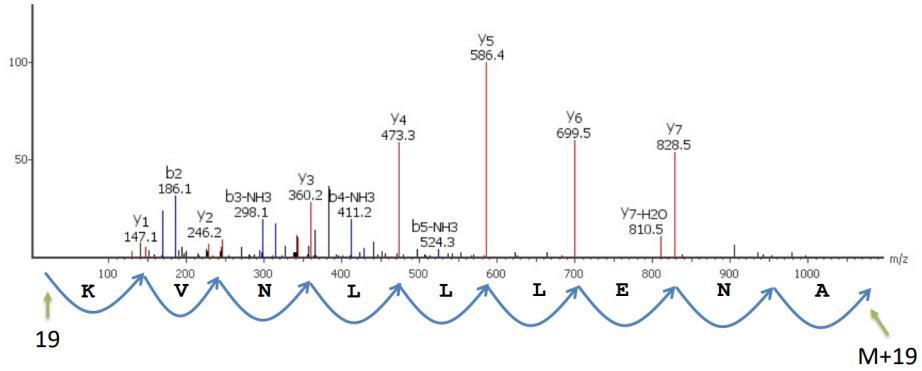


First people look for $y_1 = 147$, because they know the last amino acid must be either P or R. Then people start to look for another peak that is high and also the difference between y_1 and the other peak is equal to one of 20 amino acids. However, we might have multiple choices for y_2 or no choices.



We cannot afford exhaustive search for all combinations. For example, given length-30 peptides, it takes 20^{30}. Given the speed is 1 billion peptides per sec, it takes over 10^{22} years to search. Let's develop an efficient algorithm instead.

10.2 Dynamic Programming



where M is total residue mass. A sequence corresponds to a path that connects the y-ions. Note the reversed sequence in the path because of the use of y-ions.

Let $f(m)$ be the ion matching score at m/z value m. For example, if the log relative intensity score is used, then

$$f(m) = \begin{cases} \log_{10} 100x, & \text{if there is a peak nearby } m \text{ with relative intensity } x > 0.01 \\ 0, & \text{otherwise} \end{cases}$$

where "nearby" means error $\leq$ error tolerance. Note that this score can be precalculated for each m.

Then path score is the total of $f(m)$ for all y-ions. For example, score of path is $\sum_i f(m_i)$. Then De novo sequencing is to find a path with maximum score.

What's our input? Given spectrum, $f(m)$ can be precomputed for each m/z value m, without knowing the peptide sequence. Also, the total residue mass M can be computed from the precursor m/z and charge state.

$$\text{precursor m/z} = \frac{M + m(\text{H}_2\text{O}) + 1.007 \times z}{z}$$

Thus, we assume we have $f(m)$ and M in our input.

Equivalently, we are given an array $f(m)$ and M, and we want to find a path from 19 to $M + 19$, such that

- Each step length is an amino acid residue mass.
- The total of the scores in the visited cells is maximized.

Let $D[m]$ be the maximum score a path from 19 to m can achieve. There are two possibilities. One is empty path. If the path is not empty, assume a is the last amino acid, then $D[m] = D[m - m(a)] + f(m)$. Thus, $D[m] = f(m) + \max_a D[m - m(a)]$ where a is all possible letters.

Algorithm 13: Dynamic programming for De novo peptide sequencing

```
1 D[19] ← 0 and all other cells to be −∞. // D[< 0] = −∞
2 for m ← 20..M + 19 do
3     D[m] = f(m) + max_a D[m − m(a)]
```

The time complexity is $O(M)$.

The best sequence can be retrieved by a backtracking process by repetitively computing the last amino acid a that maximizes the recurrence relation. Time complexity is O(length of peptide).

As usual, the basic algorithm looks simple. But the reality is more difficult. In high resolution data,

nominal mass is not good enough. Error up to ± 0.5Da. We can multiple each mass (including both amino acid mass and peak m/z) with 1000 and round to integer. For example, $123.4567 \Rightarrow 123456.7 \Rightarrow 123457$. The rounding error is then limited to ± 0.0005Da. However, this increases the complexity.

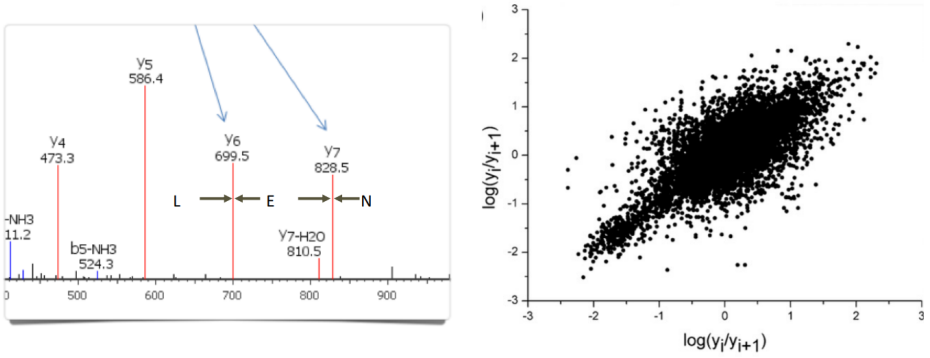
Variable PTM does not cause major speed slow down for de novo sequencing algorithms. Instead of trying 20 regular amino acids in the maximization, the algorithm simply tries all modified amino acids too. The time complexity is increased by a constant factor. (Compare to the exponential growth in database search approach). However, since the solution space is larger when many variable PTMs are allowed, the accuracy of the algorithm is reduced.

In database search, we can include other ions in the score. But here, we need to be careful because dynamic programming becomes complicated for b-ions: not guarantee an optimal solution.

For below average spectrum, as much as 50% error rate! Mostly mass gap error. If the spectrum is not perfect, for example: `ESGPA|L|V|K|PTQ|T`. We don't see fragmentation everywhere. Then there's no way to distinguish `PTQ` with `TPQ` or `QTP`. So error source could be spectrum quality or inaccurate scoring function.

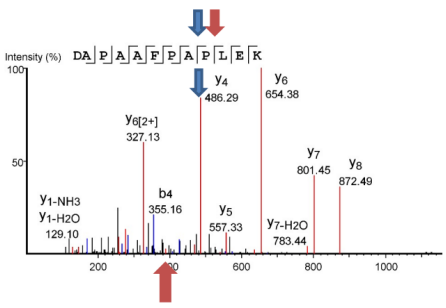
To solve this, we can make use of partially correct de novo sequence tags. Or improve the scoring function. Next let us examine one such effort, the Novor software.

10.3 Novor



For example, this spectrum might represent `TSLN|E|LQK...` where first `|` is y_7 and second is y_6. The abundance of y_6 and y_7 is determined by the easiness of peptide fragmentation on these two locations. Then the right diagram shows the ratio for 3-mer: x-axis is the ratio in one of the occurrences, and y-axis is the ratio in a different occurrence/peptide. As the points are scattered around the diagonal line, we observe that the neighbouring 3 amino acids approximately determine the peak intensity.

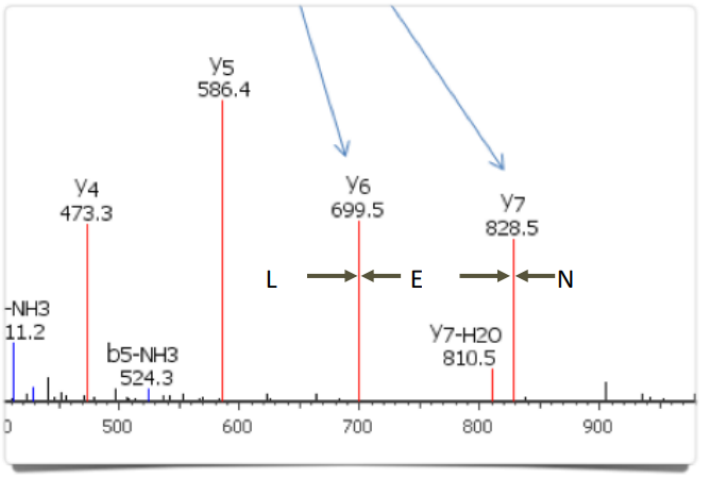
Example: Proline (P)



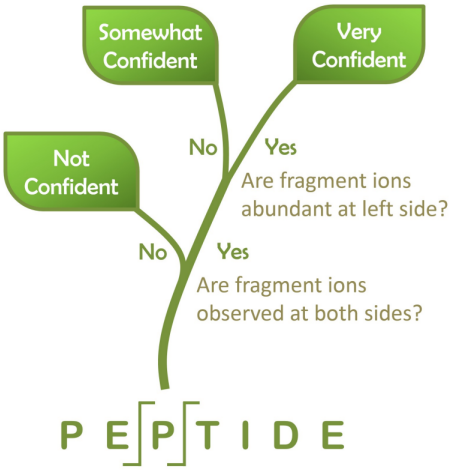
Fragmentation left to `P` is abundant, and fragmentation right to `P` is suppressed. Blue arrow shows left fragmentation and red shows right fragmentation. So we see right gives 3 ions (bottom red

arrow), and left gives y_4. Thus we cannot just reward high abundance peaks.

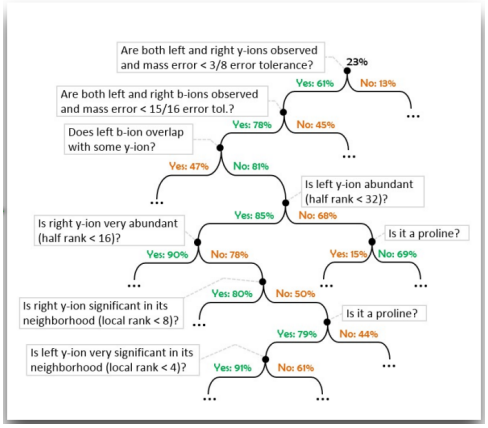
Most software's scoring function prefers more abundant peaks. Pro enhances fragmentation at left, and reduces at right.



Now we can have some scoring features. Pr(E is correct) predicted by features such as mass error; intensity of y_6 and y_7; intensity ratio y_6/y_7; L, E and N; and many others.



In Novor, it uses a decision tree. Then Novor uses NIST Spectrum Library (340,00 spectra) to do decision tree learning.



It has 169 features, 14,000 internal nodes, average depth 18.4.

Decision tree has lots of benefits:

- Allows to use of a large number of scoring features (Mass error, sequence pattern, all ion types, intensity, etc. 169 features)
- Learn a large number of rules (14,000 branching nodes)
- Each evaluation is fast. Path from root to a leaf average length = 18. Only most important features are examined according to situation.
- The result is interpretable.

A peptide's score is the sum of amino acid confidence score. Algorithm computes a peptide to maximize this score.

In Ma, B. (2015). Novor: *Real-Time Peptide de Novo Sequencing Software*, we see the Novor's result is better than PEAKS, in terms of precision and recall, and speed on a laptop PC. Novor is first and only real-time de novo sequencing software.

10.4 Basic idea of De Novo

We need to ensemble all pieces or peptides to get the protein sequence. The idea is

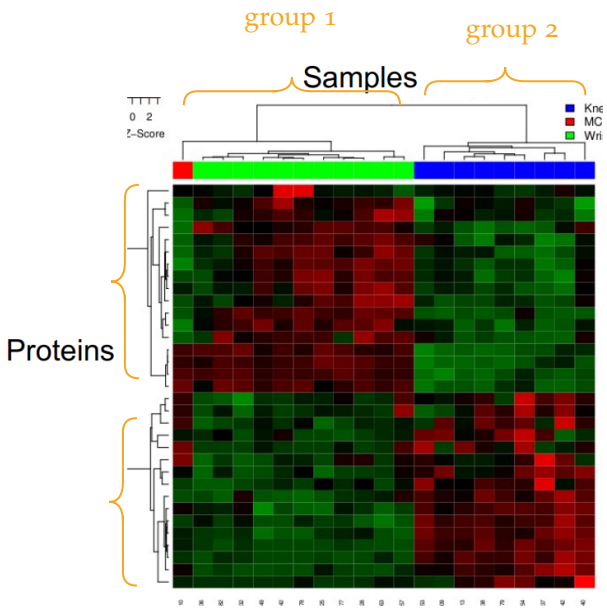
1. Digest protein with different enzymes.
2. De novo sequence each peptide.
3. Assemble overlapping peptides to derive the protein sequence.

Many de novo sequencing programs: Sherenga (1999), Lutefisk (2001), PEAKS (2003), PepNovo (2005), Novor (2015), DeepNovo (2017)

11

Peptide Quantification with Mass Spectrometry

Why do we need quantification? We want to study difference between two different cohorts.



For example, we get both samples from black group and white group and see which proteins are over-expressed or up-regulated. Then we can understand what are possible consequences of that disease. Here we compare protein quantity changes across two or more conditions to identify biomarkers. Then from the result of experiment, we can find those proteins with statistically significant different between groups.

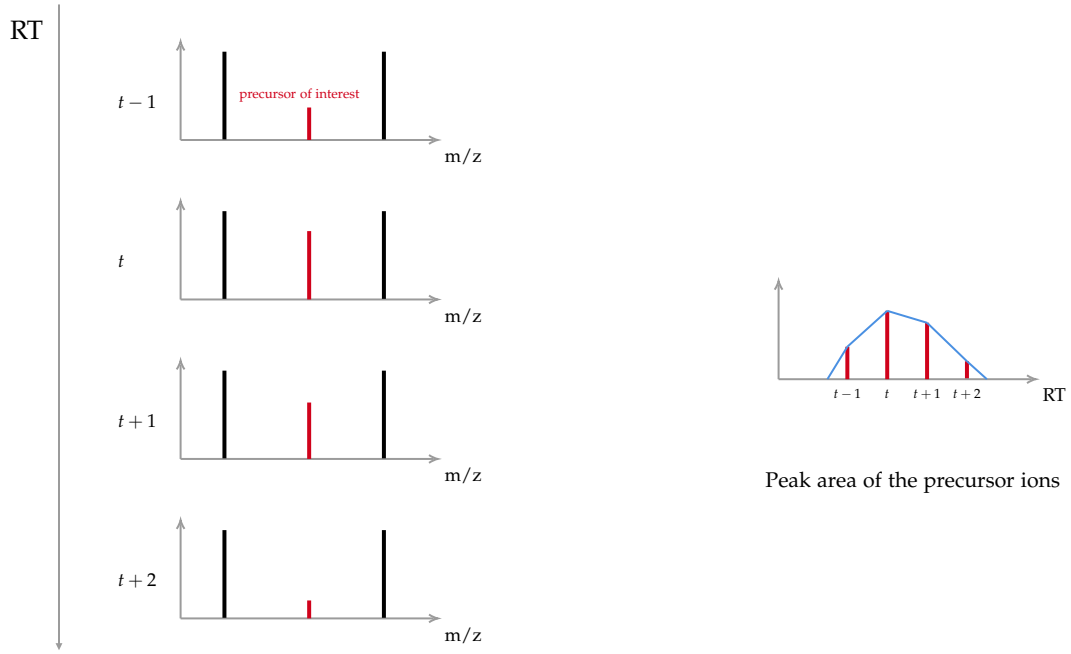
How do we do the quantification? Traditionally, we can use antibodies or other ways. Today with mass spectrometry, it provides a high throughput way to measure the protein. However, mass spectrometry is *not* a quantitative machine. Given different peptides, they can have *different* efficiency for producing MS signals, because of peptide loss in the whole process, ionization efficiency, and so on. Signal intensity is the amount of ions arriving the detection and **not** the peptide quantity.

However, if we only look at one peptide, and measure twice, under the same conditions. The rate of loss would be approximately the same between two experiments for the same peptide. This provides

a way to study the **ratio** of the changes. Namely, we can study the ratio (relative quantity) of same peptide between two MS runs.

11.1 Label Free Method

We separate all peptides on the return time dimension. At different time, we do survey scan. We notice that peptide does not come off at a single time point, but in a time interval. From MS1 scans at different retention time, we get a curve. The area below the curve is *peak area* the precursor ions, which becomes the indicator of the quantity of that peptide. This method allows us to calculate the peak area of multiple precursor ions in one experiment.



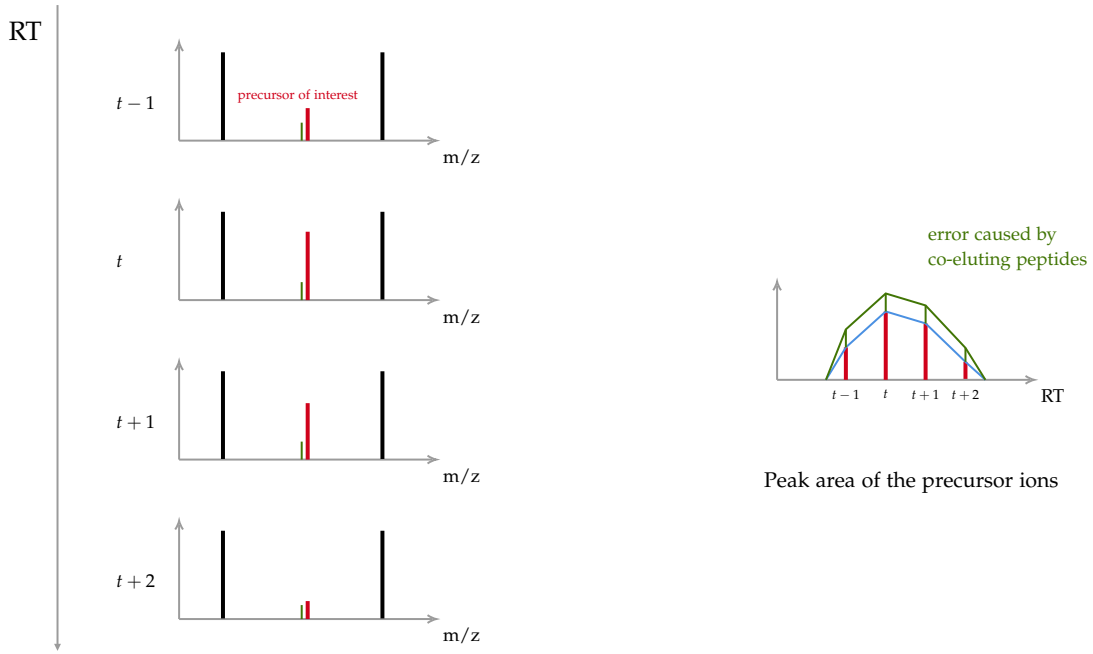
Peak area of the precursor ions

Now suppose we have two samples, and we want to find the relative quantity change. We then calculate the ratio between two areas. Then the ratio between peak areas indicate the ratio between peptide quantity. We focus more on the *concentration* change, not the *amount* change. Thus normalization may be needed to reflect the sample loading deviation.

In Label Free Method,

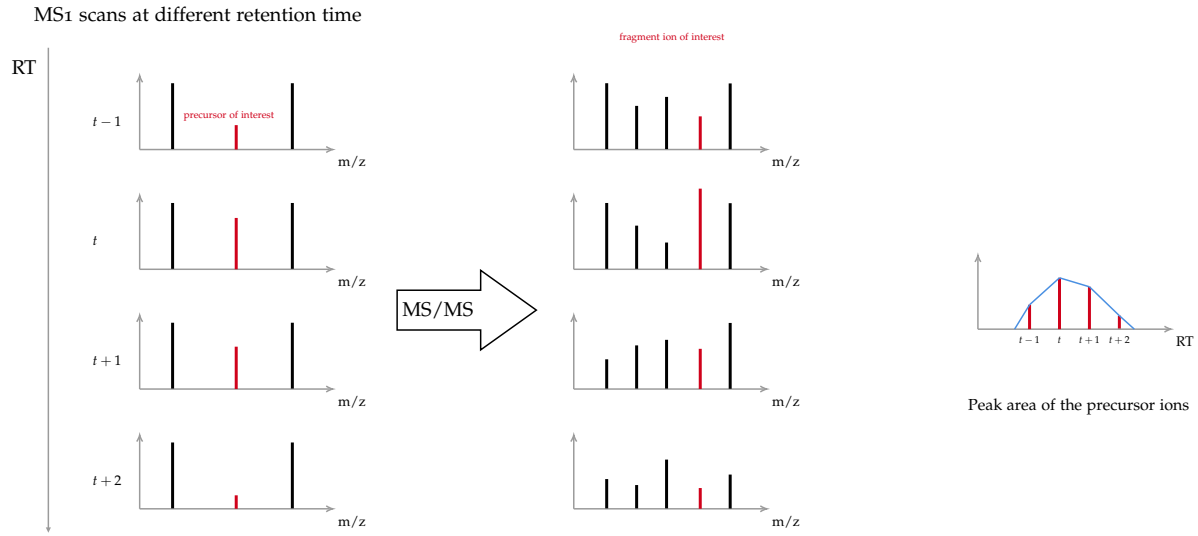
- In DDA (Data-dependent acquisition) mass spec, MS1 scans are periodically collected for precursors. MS2 scans are acquired to identify the peptides for the precursors.
- DDA on two protein samples. Identify the peptides. For each peptide identified in both samples, use their precursor peak area ratio to measure the peptide quantity change. Then we will have a list of peptides both identified and quantified.
- Often a housekeeping protein (or a spiked-in protein) is used as the normalization factor to correct the sample loading error.

However, there's problem with label free method. Suppose we have another peptide peak, which is very close by the precursor of interest. We call it **co-eluting peptide**.



So peak area is not 100% accurate. Especially when the peak area of the precursor ions is small, then any interference from other peptides will change the ratio significantly.

11.2 SRM (Selected Reaction Monitoring) Method

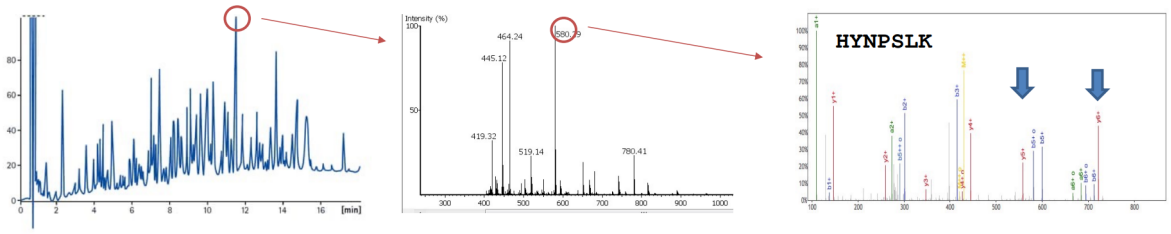


Instead of plotting the peak area of precursor directly, we program mass spectrometry to keep monitoring the precursor. Then if we see the precursor in MS1 scan, we also do MS2 scan as well, where we focus on that precursor and fragment it. Then the chance that the other co-eluting peptides produce the same fragment ions overlapping with the fragment ion of interest is low. Then we avoid interference as much as possible.

To summarize, we select precursor, fragment it, and monitor fragment ions. This provides more accurate way to do quantification.

Label free quantification with MS1 can be easily interfered by co-eluting peptides with the same m/z. SRM is more specific. Interference signal is only from co-eluting peptides with the same precursor m/z d the same fragment ion m/z. But it quantifies fewer peptides because it requires many MS2

scans to quantify one peptide. But this increases sensitivity too. But it requires the prior knowledge about which precursor and fragment ions to monitor. If multiple fragment ions are monitored simultaneously, it's called the PRM (Parallel reaction monitoring). SRM and PRM are sometimes called the **targeted method**.



LC separates the peptides according to retention time (RT).

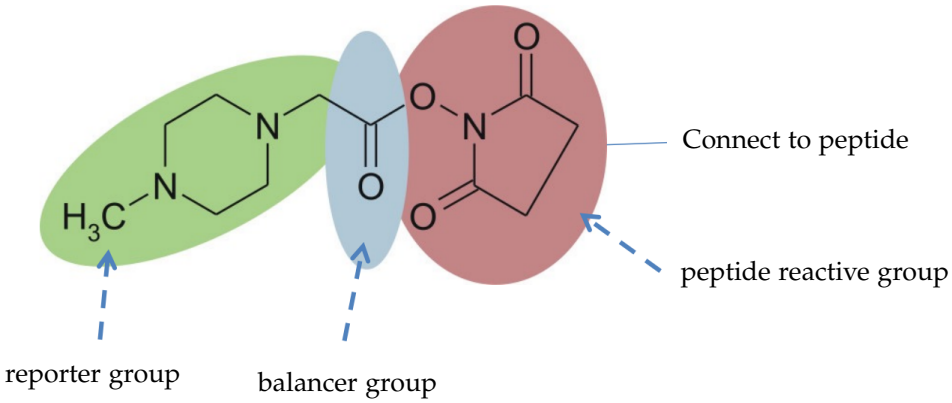
Co-eluting peptides are further separated by their mass-to-charge (m/z) in MS.

Peptides with both the same RT and m/z are further fragmented into fragment ions. Fragment ions specific to the target peptide are used to determine the quantity.

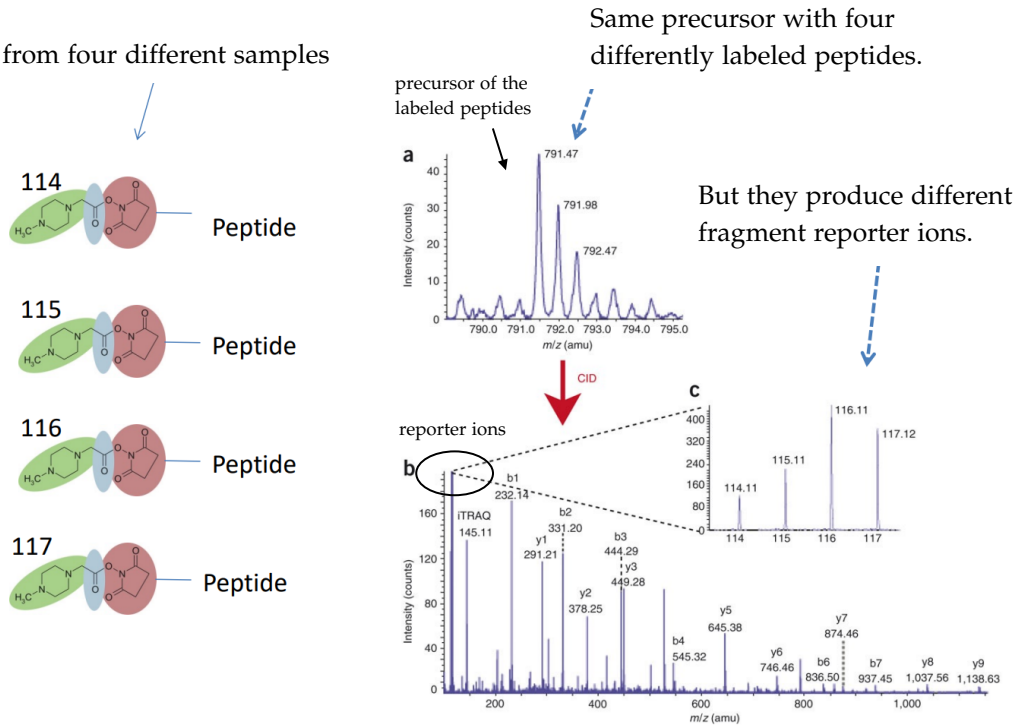
Such 3-tier separation ensures the specificity of the peptide quantification method.

11.3 Stable Isotope Labeling Method

Atoms have isotopes. E.g. C has C-12 and C-13. They differ by 1 Da. The same peptide from two samples are labeled with different isotopic reagents, and mixed together for a single MS run. Use the mass difference to known which peptide is from which sample. Many proposals: ICAT, iTRAQ, SILAC. We use iTRAQ as an example.



In labeling method, we dye a labeling region, which puts a label on each peptide. The four available tags of identical overall mass vary in their stable isotope compositions such that the reporter group has a mass of 114-117 Da and the balancer of 28-31 Da. The fragmentation site between the balancer and the reporter group is responsible for the generation of the reporter ions in the region of 114-117 m/z.



Now we have four versions. So we have four different samples, then we react and add this label to the peptide. Because they have different labels, the reporter groups will be fragmented as well. In b, these reporter ions indicate the composition of four different labeled peptides. Then the relative ratio between these four reporter ions becomes the relative quantity of the peptides in the four different samples.

For labeling based method:

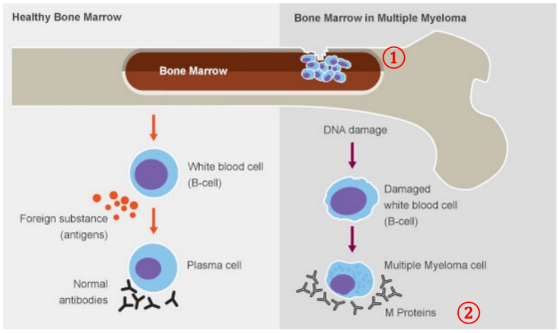
- **Pro**: Samples are measured in the *same* MS run. Therefore, more accurate, no run-to-run bias.
- **Cons**: Labeling efficiency, error and cost.

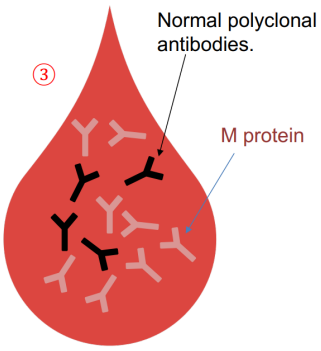
11.4 Multiple Myeloma (MM)

This (多发性骨髓瘤) is the cancer derived from the plasma cells[1] (therefore a blood cancer). It grows in bone marrow (骨髓). 30,000 new patients each year in US. Lifetime risk 1 in 143. It used to be a serious problem, but now there are effective treatments. Once it is treated, almost all of the patients will have a relapse (复发) a few years later. Patients in remission (缓解) requires *periodic* bone marrow aspiration (骨髓穿刺) to detect cancer. We want to treat relapse early.

We want a blood test, rather than bone marrow exploration.

[1] 浆细胞(Plasma cell), 亦称为效应B细胞(effector B cell)

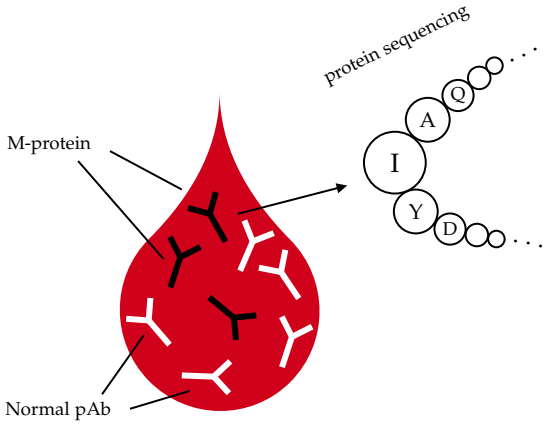




1. Myeloma resides in bone marrow.
2. Myeloma produces a monoclonal (单克隆) antibody, known as the M protein. (side-product of the cancer, not the problem)
3. M protein circulates in blood, making it a perfect marker for non-invasive measurement.

M-protein is **hard** to detect. M-protein (usually) is just a usual antibody. It is just one of many thousands of different clones of antibodies circulating in blood. These antibodies differ at their sequences, but are very similar in shapes. With some of the traditional methods, M-protein can only be realized when it is highly abundant - more than the total of all other antibodies. After treatment, it becomes too low to be detected. Such MRD (minimal residual disease) causes relapse.

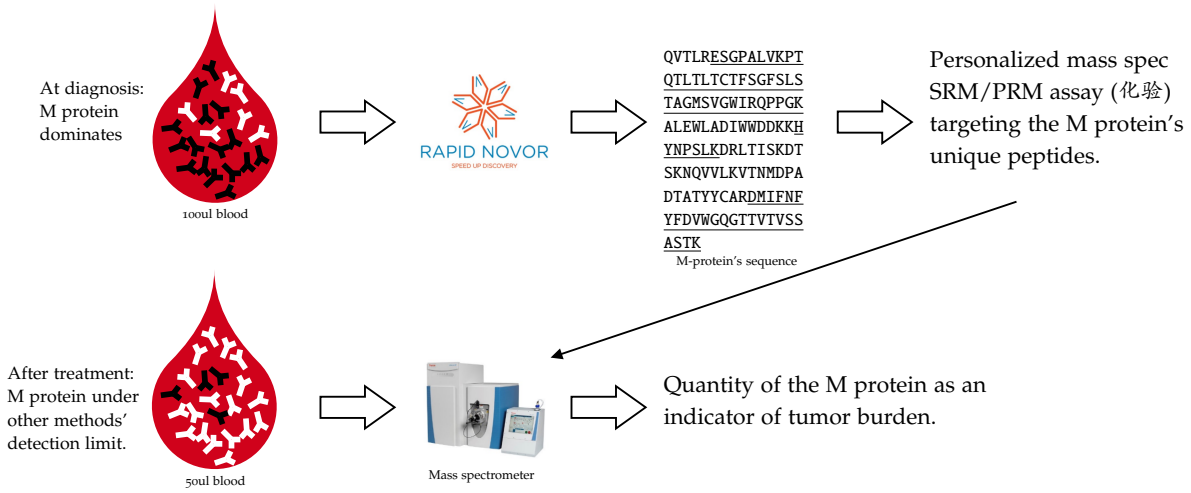
Then people come up the solution with MS spectrometry. Although M-protein looks like a normal antibody, we know actual clone is different in every sequence. We can read out the animo acid sequence of each protein, then we can tell which one is from the cancer.



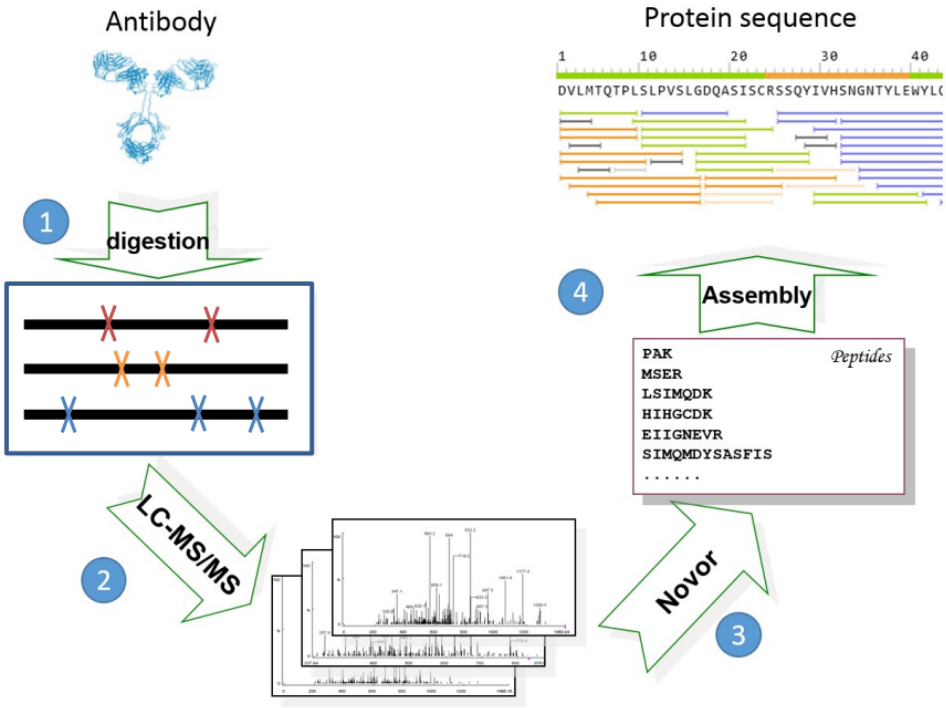
1. At diagnosis: Determine the amino acid sequence of the M-protein, pick a characteristic peptide that is unique to the M-protein.
2. Follow up: Measure the characteristic peptide with targeted mass spectrometry experiment.

So after the treatment, we get a sequence for M-protein. Then do enzymatic digestion, we get peptides specific to M-protein. Then we monitor the quantity of that peptide using SRM.

Here is Rapid Novor's Platform:



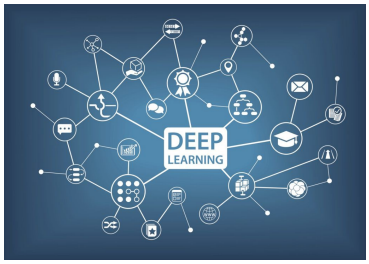
Here is Protein Sequencing with REmAb^{TM}:



1. Digest with multiple enzymes
2. LC-MS/MS
3. Peptide de novo sequencing
4. Sequence assembly

EasyM - Mission Accomplished

- Blood based. No need for bone marrow aspiration. It's more safer, thus can do more frequently.
- 1000 times more sensitive than other blood based method
- Detect relapse about half year earlier than conventional methods.



Part III:

Sequence Annotation and Machine Learning

https://towardsdatascience.com/
does-deep-learning-really-require-big-data-no-13890b014ded

12

Spectrum Prediction with DNN

12.1 Basics

We've briefly used machine learning twice now: Use a decision tree to score an amino acid in a de novo sequence; Combine multiple score features in database search.

We combine multiple features together, with a function, and produce output: e.g., $y = f(w_1x_1 + w_2x_2 + b)$. The output will normally be processed through an activation function, e.g., $y = \frac{1}{1+e^{-z}}$ (sigmoid). Train the coefficients w and b to maximize the separation of true and false data points. For training, a cost function is defined and there are optimization algorithms to minimize the cost. Under certain cost function and sigmoid activation, this is equivalent to the logistic regression. This is done by **neuron**.

For **neural network**, we simply put neurons together. A neural network just combines many neurons together to fit a more complex nonlinear function. Often these neurons are organized in layers. The algorithm for training is usually the so-called backpropagation algorithm. Main idea behind back-propagation is gradient descent implemented in a nice way.

Some training terms:

- Known correct (input, output) pairs.
- A cost function.
- Adjust the parameters gradually to reduce the total cost over all training data: gradient descent.
- Training, validation and testing data are separate.
- Training: to learn parameters
- Validation: control the training process to avoid overfitting
- Testing: test the real performance after training

Learning and prediction:

- Prepare training, validation, and testing data.
- Specify the network structure in a deep learning framework (e.g. Tensorflow)
- Learn parameters with the training data. Control the learning process with the validation data.

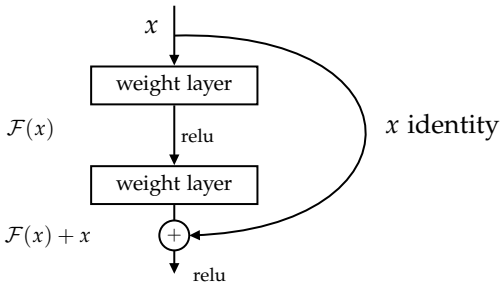
- Test the performance on the testing data ONLY after the training is done and parameters are fixed.
- Use the trained parameters to make prediction for future data.

Deep Neural Network is just a neural network with MANY layers. MANY coefficients (parameters) to train. Require new training algorithms to both learn fast and avoid overfitting. GPU, big data, and new learning algorithm contribute to the development.

People have a better idea: **Convolutional Neural Network**. When the input has *spatial similarities*, one can reuse the coefficients. A "filter" can be seen as a neutron on a local region of the input (kernel). By sliding the filter over the input array, one can produce the output array. This is called **convolution**. Number of coefficients is reduced significantly. Reduce overfitting. Can afford more layers. Capture the local spatial correlation by focusing on each local region.

Similar to CNN, deep learning researchers developed other structures to connect between layers that can boost up the learning performance. Some of these common structures can be reused as building blocks in other learning tasks. Just like programming patterns. There are deep learning frameworks that allow the users to assemble the building blocks in a flexible way, and provide the learning and prediction support: Tensorflow, Pytorch etc.

Another commonly used building block is the residual block: **ResNet (Residual Neural Network)**. Intuitively, each additional residual block uses one or more layers to predict the difference between x and the desired output. This approach has proven to both improve the learning performance (both learning speed and accuracy). Residual blocks are supported by most of the popular deep learning frameworks.



This requires dimensions of $\mathcal{F}(x)$ and x are the same.

12.2 Spectrum Prediction

Now we use it to do spectrum prediction. We start with a peptide sequence, then with MS/MS, we will have MS2 spectrum, which will identify the peptide sequence. These are all bioinformatics effort. Then **spectrum prediction** is to take the sequence, and predict the spectrum, so that it matches the experimental spectrum. This allows us to confirm the peptide sequence is correct.

Let's start with a simpler task to learn the basics: To predict whether a peptide is detectable in a DDA experiment. This is called **peptide detectability**. Not every peptide of the proteins will be identified with mass spectrum, e.g., some peptide is hard to get charged.

- Input: A peptide sequence.
- Output: 0 or 1. (or a probability that the peptide is detected).
- We also have a lot of training data: pairs of (input, true/false).

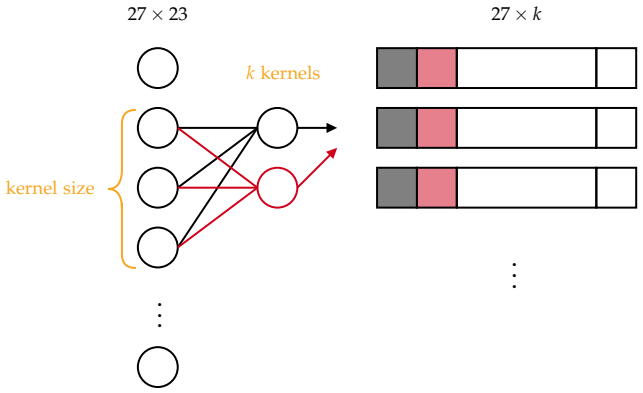
Model adopted from the DNN model in following spectrum prediction paper: *Liu et al*. Full-Spectrum

Prediction of Peptides Tandem Mass Spectra using Deep Neural Network. The paper does not predict detectability but we adopted the model anyway for studying purpose. There can be many different DNN structures for the same task.

First we need to encode a peptide. We do *one-hot encoding* of amino acids. Each amino acid is mapped to a length-20 vector. Only 1 bit is 1, the other 19 bits are 0. However, peptides have variable lengths: we limit length to be at most 25 amino acids. If less than 25, we pad with a special character. We also encode the two termini (n-term and c-term).

Thus in total $20+2+1=23$ codes are needed. One-hot requires a length-23 vector. In total $25+2=27$ positions to code. Therefore, each peptide is mapped to a 27×23 array. Note: PTM is not considered.

Then we use CNN.



Recall each amino acid is a 23-dimension 1-hot encoding. A kernel is a function that computes a value from s adjacent amino acids. Sliding the same kernel will produce a 27 dimension vector. This is the convolution. *Padding* is needed to keep the dimension 27. There are k kernels. So the output of the convolutional layer is $27 \times k$. Number of parameters is $k \times s$, which is relatively small. Activation function for the filters: sigmoid or ReLU (Rectified Linear Unit).

For next layer, we can take each channel (e.g., black channel) in the previous layer, and apply k' kernels on it. Thus $k \times k'$ kernels, producing $k \times k'$ channels. Three practices are often used to avoid exponential expansion of number of parameters:

- $k' = 1$, where we don't increase number of channels.
- Stride > 1. Stride denotes how many steps we are moving in each steps in convolution.
- Pooling. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. For example, max pooling.

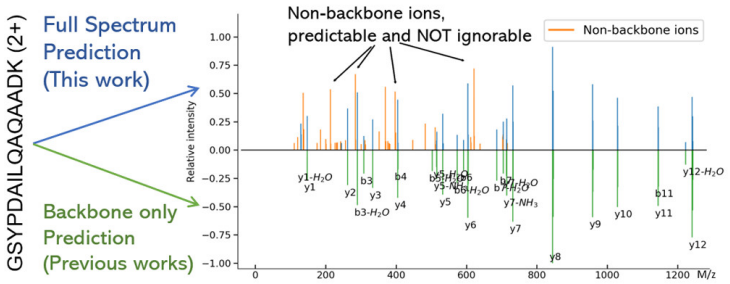
With controlling of exponential growth, we can keep adding more and more layers.

Then the output of the last convolutional layer ($n \times k$) is connected to a few fully connected (dense) layers. At last, the output layer computes the output. Output layer can contain one or more neurons - depending on the actual output format. The fully connected layers have the largest number of parameters to learn. Most of the time, in the middle, ReLU is used as activation function, due to its efficiency. At the end, sigmoid is normally done.

Now we have the architecture for handwriting digit recognition (not showing here...). We then can modify it to predict the spectrum. There are other ions (not y and b-ion). Should these minor ions (not y and b) be predicted? or only y-ions and b-ions?

There are two lines of research: To predict only the main fragment ions (e.g. b and y ions); to predict the full spectrum. Both have been done in literature. We study the full spectrum prediction in the

following paper (previously mentioned).

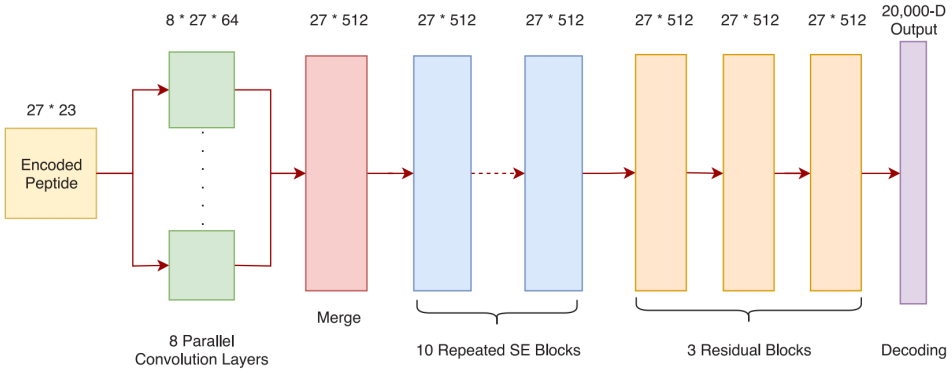


From this diagram, we see that on the top, it's the prediction with other ions. Non-backbone ions are not b and y-ions.

Output format A spectrum usually has a limited m/z range. The work only predicts peaks between 180-2000. The spectrum is represented by a sparse one-dimensional (1-D) vector by binning the m/z range between 0 and 2000 with a given bin width. The value stored is the peak intensity. If we have multiple peaks within 0.1Da, it's ok to take the maximum, which happens rarely. With 0.1 Da bin width, a spectrum becomes a 20,000-dimension vector. (Note: most dimensions have value 0). The value in each bin is the *relative* intensity of the tallest peak in the bin. Prediction is to predict the values in all the bins.

Input Same as before. Maximum length 25 peptide. Padding if shorter than 25. 1-hot encoding of each amino acid, the termini, and padding. In paper, they *do not* distinguish n and c term. Additionally provide the amino acid residue mass. Thus, 23 dimension vector for each amino acid and termini.

Then this is their CNN Architecture for Spectrum Prediction.



They have kernels of size 2..9, and for each of them, they have 64 kernels. The 8 parallel convolution layers use kernel size 2 to 9, respectively. SE (Squeeze-and-Excitation) blocks are another type of commonly used DNN building blocks.

2.2M spectra with known sequences were used for training. The result is quite good.

If peptide is correct, then its predicted spectrum should match the experimental one with high similarity. Whereas the wrong peptides should have low similarity.

13

Protein Structure Prediction with DL

13.1 Four levels of protein structure

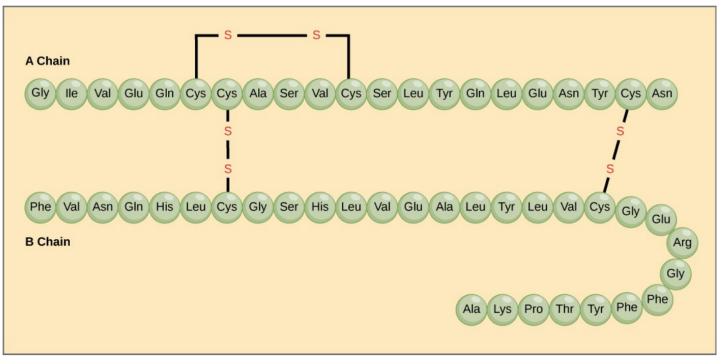
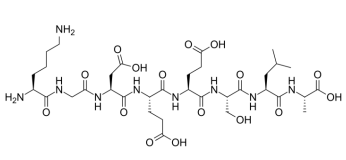
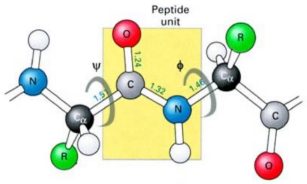


Image Credit: Khanacademy

There are different levels of protein structure. The basic level is called **primal structure**, which is just the protein sequence. Cysteine (半胱氨酸) is special, because on its side chain, there's sulfur, which is highly reactive. They are so reactive that they are connected with this **disulfide bond**.

Below is Beefy meaty peptide (delicious peptide). A torsion angle, also known as a dihedral angle, is formed by three consecutive bonds in a molecule and defined by the angle created between the two outer bonds. [1] With the sequence, all distances between atoms, and all torsion angles, we can calculate 3d-coordinates of every single atom that determines the (backbone) structure of the protein, where we might ignore the sidechain.





Nature doesn't give us random sequences. Nature needs to the protein to have certain functions so that it needs to be in certain structure. There are two common patterns for amino acids: α-helix and β-pleated sheet. It's common to form hydrogen bonds between O and H. This is called **secondary structure**, where we determine α-helix and β-pleated sheet.

[1] src: https://www3.cmbi.umcn.nl/wiki/index.php/Torsion_angle

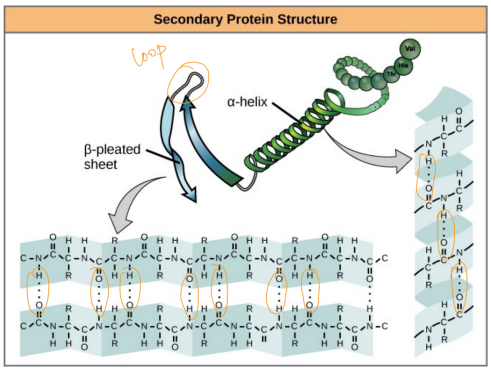
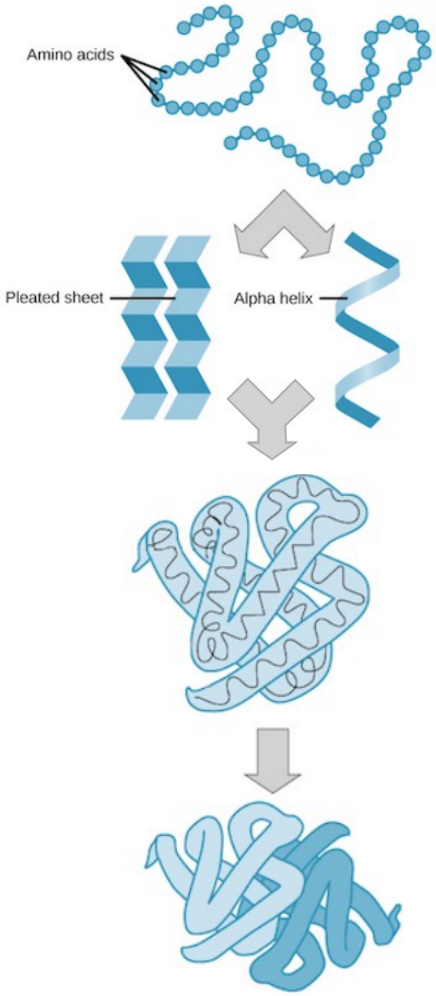


Image credit: OpenStax Biology.

Then we can go further: Tertiary and Quaternary Structures.



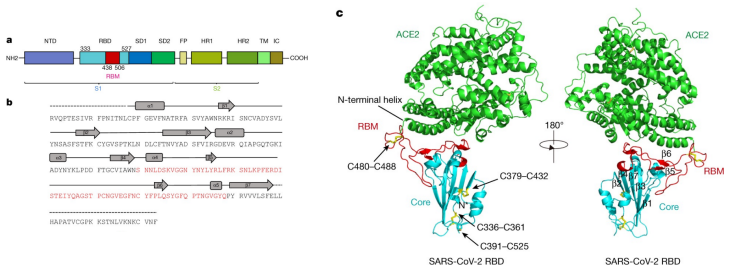
Primary protein structure
sequence of a chain of amino acids

Secondary protein structure
hydrogen bounding of the peptide backbone causes the amino acids to fold into a repeating pattern

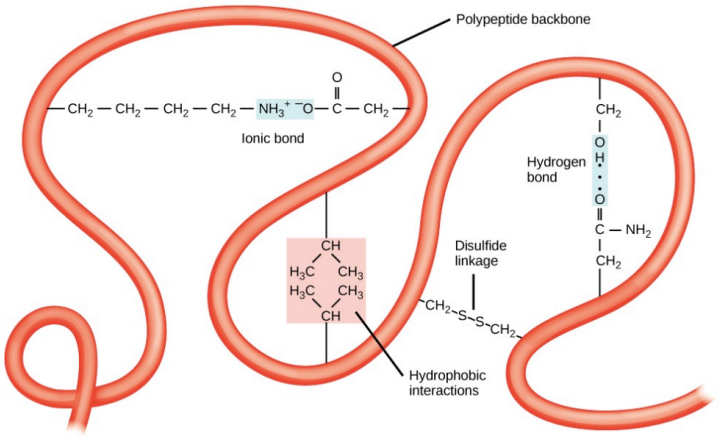
Tertiary protein structure
three-dimensional folding pattern of a protein due to side chain interactions

Quaternary protein structure
protein consisting of more than one amino acid chain

Here is SARS-COV2 Spike protein and ACE.



where RBD is receptor binding domain, a subarea of the protein.



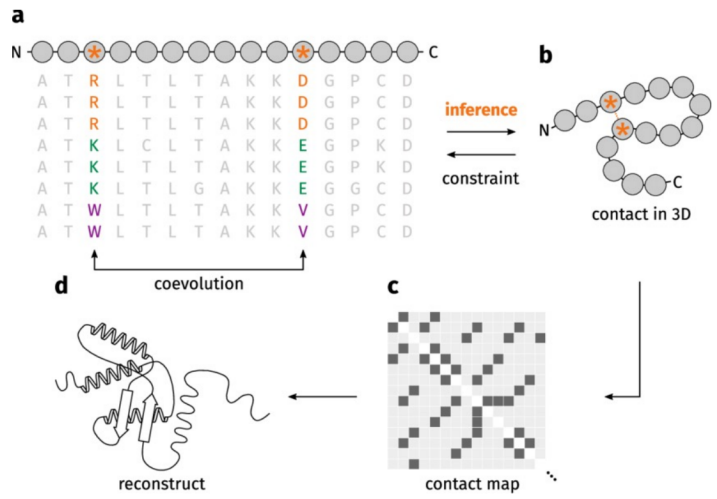
What are the forces that cause the protein to bind in a certain way? We have talked about sulfur bonds. Above we have ionic bond. This connection will reduce the overall **free energy**, which will make the structure more stable. There are other forces, like hydrophobic(疏水) interactions. It is believed that the natural structure *minimizes* the free energy.

This becomes the structure prediction problem:

- Free energy function
- Enumerate all torsion angles to minimize

This is not an easy problem. First, even for the same hydrogen bond, the free energy deduction is different. It's affected by the surroundings. Second, it's not efficient to enumerate all torsion angles.

One way to address this problem is to offset the problem of free energy estimation. Instead of the actual physical free energy for each bond, people build approximations or score functions. One method is multiple sequence alignment, and search in the database.



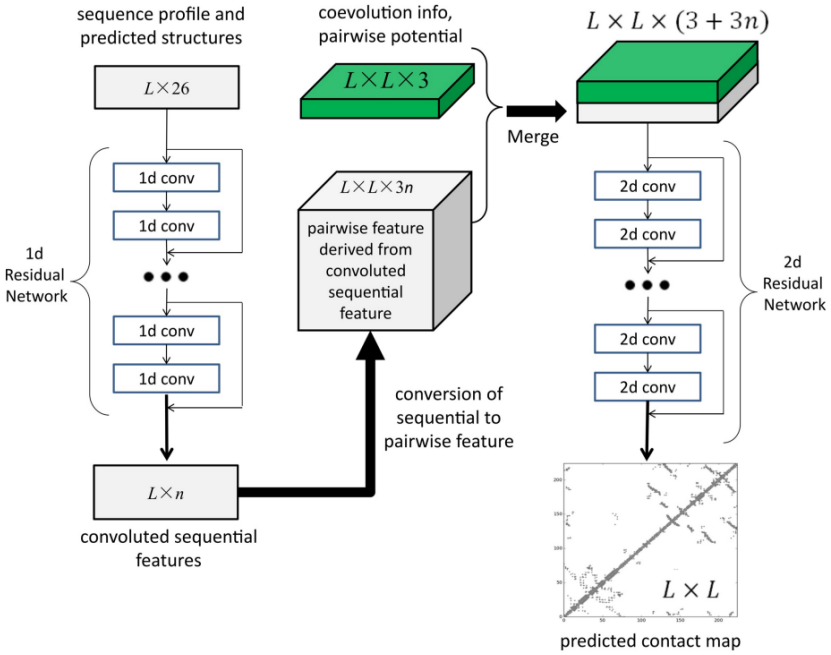
In (a), we search the gray sequence in the database, and find all other similar sequences. Then do multiple sequence alignment. Then we look at **coevolutional information**: the two starred amino acids are conserved, which is very strong signal of coevlolution. Then we have contact map: the same sequence vertically and horizontally. With contact map, we have extra constraint on our search in database. Additional information can be incorporated in the "energy" function. It's therefore just a scoring function. Image credit: https://www.nature.com/articles/s41598-019-55047-4

For the search, traditionally there are two approaches:

- Monte Carlo: Sampling of the confirmational landspace and minimize energy
- Threading: Use existing protein structures as models

13.2 Deep Learning

Above are two traditional approaches. Lately, people use DL to predict torsion angles directly, instead of doing searching.



Above is **RaptorX**. They use contact map to build more accurate protein score, then traditional search. So the improvement is on the contact map prediction. Traditionally, contact map is predicted by co-evolution. Wang S, Sun S, Li Z, Zhang R, Xu J (2017) Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model.

Then Google's deepmind started to work on this problem as well. They built **AlphaFold**: takes Jinbo Xu's approach + a gradient descent search strategy to minimize the energy, as they have lots of computing resources. Then later, they have **AlphaFold2**

1. Predict the contact map (2D representation of sequence) with neural network.
2. Followed by *torsion angle* and structure prediction with neural network.
3. New network structures, lots of novelty.

CASP (Critical Assessment of Protein Structure Prediction) is a competition, every two years. Then in Median Free-Modelling Accuracy, we see AlphaFold is better than Raptor, and AlphaFold2 is better than AlphaFold.

Structure Prediction Software

- Rosetta: David Baker, University of Washington
- RaptorX: Jinbo Xu, Toyota Technological Institute at Chicago
- Alpha Fold: DeepMind
- Many others..

Use Alpha Fold online: https://colab.research.google.com/github/deepmind/alphafold/blob/main/notebooks/AlphaFold.ipynb

Protein Structure Resources

- Structure databases
  - PDB (Protein Data Bank) https://www.rcsb.org/
  - Alpha Fold Predicted https://alphafold.ebi.ac.uk/
- Structure visualization software (one example is Pymol)

There are two remaining problems:

1. How do you predict the structure for the sequences that don't find the homologs in the database? Then not possible for multiple alignment.
2. How to design structure? If you have a structure, how to design a sequence that folds into that structure.

14

Hidden Markov Model and Gene Prediction

We will use tis model to solve gene prediction: we are given a long sequence of DNA molecule, and we want to annotate the genes. Between genes, we have non-coding areas. More precisely, from wiki: "In computational biology, gene prediction or gene finding refers to the process of identifying the regions of genomic DNA that encode genes.".

14.1 Hidden Markov Model

Hidden Markov model was first invented in speech recognition. But are widely used in many other areas including bioinformatics. It's basically an automata. An automata that has "hidden states". At each time point, it emits a symbol, and change a state with certain probability. We want to derive the hidden states by the emitted symbols.

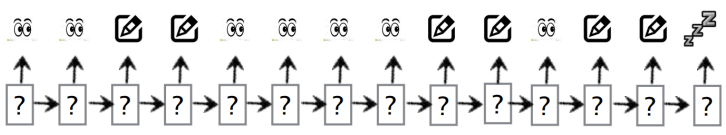
Now consider a simple **Class Example**. Think of a student in classroom. At any minute, a student is in one of 3 hidden **states** that I try to figure out:

- U: understands
- T: does not understand but tries to understand
- L: is lost completely and does not try to understand

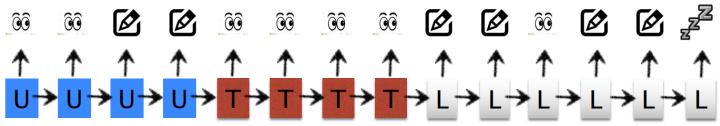
Meanwhile, the student emits one of 3 symbols that the teacher can observe:

- Look at me
- Write/Type
- Sleep

Now suppose the teacher see a student's behavior is the following in the past several minutes. What is his internal states at each minute?



And below is one possible solution:



Let's now try to formulate mathematically. We want to maximize two probabilities. One is **emission probability**. For example, when you "U", you can either look at the board or write. The other way is **transition probability**, from previous state to the current state, either can go from "U to U" or "U to T". So our task: fill all internal hidden states so that the product of all probabilities are maximized.

Typically, HMM assumes that emission probability depends only on current state; and current state only depends on previous state. We want to find the most likely path of states given the symbols (observations).

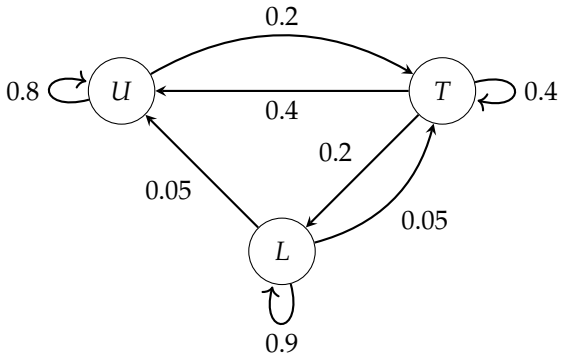
We then have two matrices (mostly by intuition):

	U	T	L
U	0.8	0.2	0
T	0.4	0.4	0.2
L	0.05	0.05	0.9

	Look	Write	Sleep
U	0.6	0.35	0.05
T	0.9	0.1	0
L	0.1	0.6	0.3

Each row adds up to 1.

For the first table, we can also visualize with a transition graph.



The input is a **sequence of symbols**: $S = S_1 S_2 \dots S_n$, and we want to find **path of states**: $P = P_1 P_2 \dots P_n$. We want to maximize $\Pr(P \mid S) = \Pr(P, S) / \Pr(S)$. Therefore, we want to maximize

$$\Pr(P, S) = \prod_i \underbrace{\Pr(P_i \mid P_{i-1})}_{\text{transition}} \underbrace{\Pr(S_i \mid P_i)}_{\text{emission}}$$

Note: To deal with the first state, we can define $\Pr(P_1 \mid P_0) = 1$ in above formula.

Now we use dynamic programming to solve HMM. Define $D[k, p]$ be the maximum probability achieved by first k states given that the last state is p:

$$D[k, p] = \max_{P[1..k]; P[k]=p} \prod_{1 \le i \le k} T[P_{i-1}, P_i] \, E[P_i, S_i]$$

Then $\max_p D[n, p]$ is the maximum probability achieved by the complete path, which is what we want to compute. It is not hard to obtain a recurrence relation

$$D[k, p] = \max_{p'} D[k-1, p'] \underbrace{\Pr(p \mid p')}_{\text{transition}} \underbrace{\Pr(S_i \mid p)}_{\text{emission}}$$

Then we have an algorithm to solve HMM.

Algorithm 14: Solving HMM

Input: $S = S_1 S_2 \dots S_n$
Output: $P = P_1 P_2 \dots P_n$

```
for every state p do
    D[1, p] ← Pr(S_1 | p)
for k ← 2..n do
    for every state p do
        D[k, p] = max_{p'} D[k − 1, p'] Pr(p | p') Pr(S_i | p)
backtrace to compute the optimal path
```

For example, given the previous transition graph and emission matrix, and $D[1, U] = 0.6, D[1, T] = 0.9, D[1, L] = 0.1$, we then find $D[2, U] = \max\{0.288, 0.216, 0.003\}$.

Note that we should **not** multiply, because soon the numbers become so small that the double precision will give you value 0. Do a logarithm and use additions instead:

$$\log D[k, p] = \max_{p'}(\log D[k-1, p'] + \log \Pr(p \mid p') + \log \Pr(S_i \mid p))$$

All of our computation depends on the transition probabilities and emission probabilities. How do we estimate these parameters?

If we have an annotated sequence with both symbols and states, then these can be trained by counting. If we do *not*, then we can start with a reasonable guess of the parameters and annotate the sequence. Then we use the annotation to train a new set of probabilities. Repeat until converge. There is some guarantee to the convergence. But does not guarantee this will converge to the right solution.

Another thing to worry about is that the training data is not enough. If the training data include no cases of a particular emission from a particular state, then its probability will be 0 in this model. That's no good. So we add pseudocounts to make the probabilities not zero when an event should be able to happen. For example, put a small number in the zero entry.

We can also expand basic HMM model to **high order HMM**. Let's still consider the classroom example. Previously, emission only depends on the current state. Sometimes emission symbol depends on the previous symbols. The emission of a symbol should not only depend on current state, but sometimes also the previous symbol. For example, sleeping at previous moment leads to a higher probability of sleeping now. To accommodate the correlation between the adjacent symbols, the emission matrix needs to be expanded. The emission matrix becomes $\Pr(S_i \mid P_i, S_{i-1})$. This is **first order HMM**.

Now the probability

$$\Pr(P, S) = \prod_i \Pr(P_i \mid P_{i-1}) \Pr(S_i \mid P_i, S_{i-1})$$

To find the path P to maximize, we let $D[k, p]$ be the maximum probability obtained by the first k states ending at p. We can obtain the following recurrence relation similarly as before.

$$D[k, p] = \max_{p'} D[k-1, p'] \Pr(p \mid p') \Pr(S_i \mid p, S_{i-1})$$

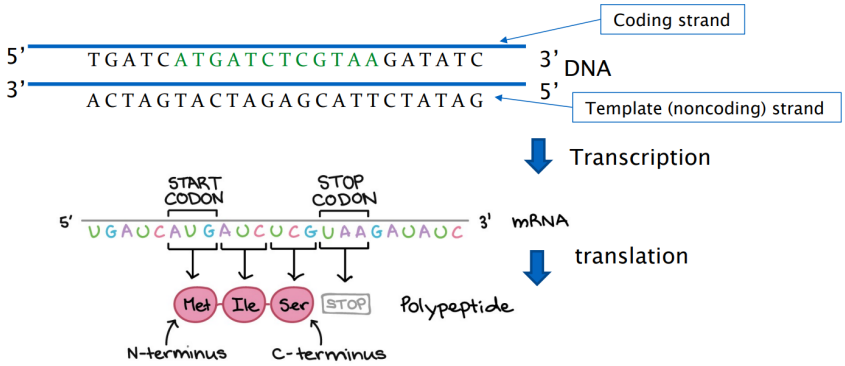
We can still do dynamic programming.

To generalize, we can let the current emission depend on the current state, and previous k symbols. Then this is called the ***k*-th order HMM**. Solving such a HMM is similar as before. Running time not changed. The only difficulty is the parameter training because the emission matrix has many more parameters for larger k.

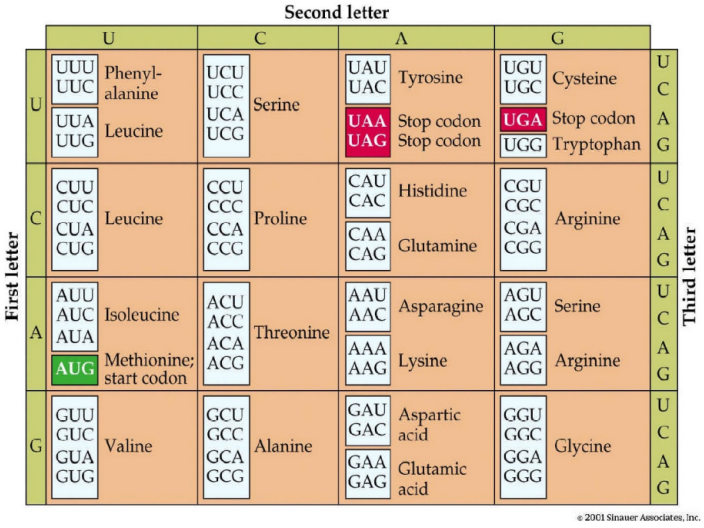
14.2 Trivial Gene Finding

The prokaryotes (singular prokaryote, 原核生物) are a group of organisms that lack a cell nucleus (= karyon). The opposite is the eukaryotes (真核生物). Prokaryote genes do not have introns (内含子). So their genes is a linear structure. Intron video: https://www.youtube.com/watch?v=o0BQJbLNYSg

Here is the process from gene to protein (in Prokaryotes):



where transcription means 转录 and translation means 翻译.



The genetic code[1] is well studied. Here is a genetic table. Every three letters matches to a single amino acid. A codon is a trinucleotide[2] sequence of DNA or RNA that corresponds to a specific amino acid.

With the genetic table, we can build a *trivial gene finder*. We know that every gene will start with *start codon* ATG and stop at three *stop codons* TAG, TGA, TAA. So we look for **Open Reading Frame** (ORF) is a substring that:

- starts with a start codon
- ends with a stop codon
- no stop codon in the middle

If ORF is long, then likely it is a gene or a part of a gene. Why? Basically we have two hypotheses.

[1] 遗传密码又称遗传编码，是遗传信息的传递规则

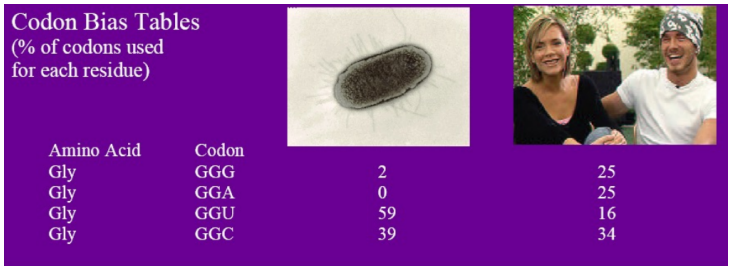
[2] Trinucleotide, or triplet, repeats are 3 nucleotides consecutively repeated (eg, CGG CGG CGG CGG CGG) within a region of DNA.

One is that this is a gene, so everything looks natural. The other is that this is not a gene, so everything is random. Note that probabilities of stop codon is $\frac{3}{64}$. Given a long sequence with many codons, the probability that there is at least one stop codon in the middle is high.

We have **codon bias**. A codon XYZ occurs with different frequencies in coding regions and noncoding regions:

- different amino acids have different freq.
- different codons for the same amino acid have diff. freq.
- in random regions $\approx p(X) \times p(Y) \times p(Z)$.

Here is codon bias tables:



http://www.kazusa.or.jp/codon/ is a public database where we can find codon bias.

14.3 Better Gene Finder

Now with this table, we can calculate the probabilities to find whether ORF is random or a gene: $\Pr(seq \mid \text{gene}), \Pr(seq \mid \text{random})$. Then we can use this additional information from codon bias to find trivial gene finder. More specifically, We can use the log likelihood ratio score to evaluate each ORF. Each codon XYZ contributes score

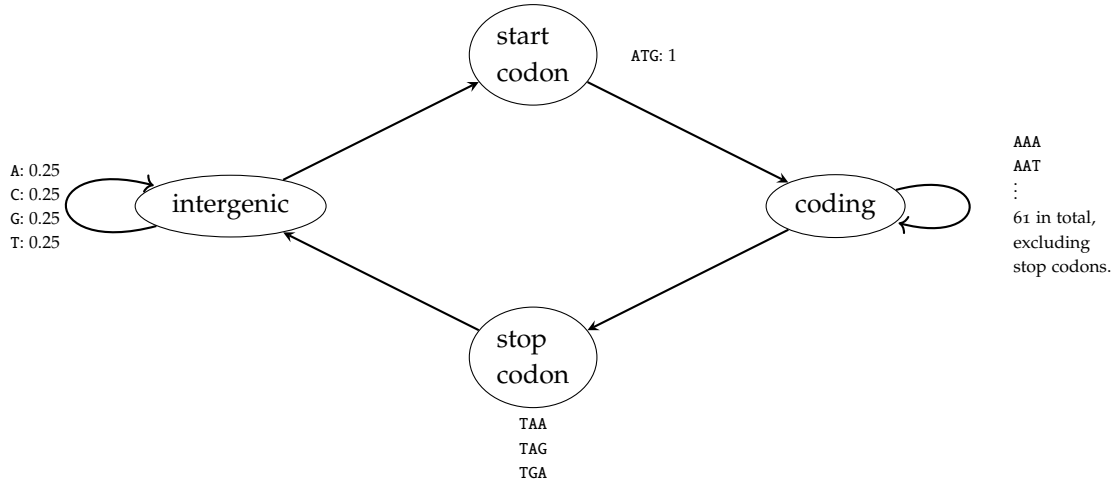
$$\log \frac{P(XYZ)}{P(X)P(Y)P(Z)}$$

An ORF is predicted as a gene if the sum of codon score is above a threshold. This is better. But it does not catch the correlation between adjacent codons.

14.4 HMM Gene Prediction

We have used HMM in the classroom example to catch correlations between adjacent events. This can be used to model gene prediction. For example:

- Symbols: Nucleotide bases.
- (Hidden) states: start codon, stop codon, coding, non-coding (intergenic).

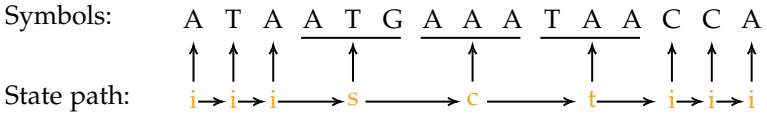


where 0.25 means that in the intergenic region, emission of `A, C, G, Y` is of the same probability. And for coding state, we can assume emission probability is $\frac{1}{61}$ for each codon, or can use codon bias table. For stop codons, similarly we can use $\frac{1}{3}$ or codon bias table.

Then we can make reasonable transition probability. Assume every gene has about 300 codons, then

$$\begin{aligned}
\Pr(start \to coding) &= \Pr(stop \to intergenic) = 1 \\
\Pr(coding \to coding) &= \frac{299}{300} \\
\Pr(coding \to stop) &= \frac{1}{300} \\
\Pr(intergenic \to intergenic) &= \frac{99}{100} \\
\Pr(intergenic \to start) &= \frac{1}{100}
\end{aligned}$$

And it's better to train these parameters.



Top sequence is the genome and bottom sequence is the annotation of the genome. Annotated the sequence with most probable path of states:

- start codon
- coding
- intergenic
- stop codon

This provides a reasonable answer to gene prediction. A *difference* here: emission is not fixed length. But this does not forbid us from solving it with dynamic programming.

We define $D[k, p]$ be the max probability achieved by first k symbols for a path with the last state being p. For the recurrence relation, there are several cases:

1. For $p =$ intergenic. Assume previous state is p'. Then the emission is of length 1, we have

$$D[k, p] = \max_{p'} D[k-1, p'] \times \underbrace{\Pr(p \mid p')}_{\text{transition}} \times \underbrace{\Pr(S_k \mid p)}_{\text{emission}}$$

2. For $p =$ start, coding, or stop, emission is of length 3, we have

$$D[k,p] = \max_{p'} D[k-3,p'] \times \underbrace{\Pr(p \mid p')}_{\text{transition}} \times \underbrace{\Pr(S_{k-2}S_{k-1}S_k \mid p)}_{\text{emission}}$$

Once the recurrence relation is obtained. It is straightforward to work out a dynamic programming algorithm, as well as backtracking. This is very easy to implement. If desired, one can also use a higher order HMM. Parameter training must be done carefully.

14.5 Promoters

Besides the codon bias that can be captured by HMM, there are other signals in a gene structure that can be employed by a gene prediction program. E.g. the **promoter**[3] of a gene is a region of DNA sequence located near the start codon. For example,

```
<-- upstream                                                    downstream -->
5'-XXXXPPPPPPXXXXXXXXXPPPPPPXXXXGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGXXXX-3'
       -35            -10                        Gene to be translated
```

Then we have at `-10` and `-35`, the sequence is probability described by the following table:

-10:	T	A	T	A	A	T
	77%	76%	60%	61%	56%	82%
-35:	T	T	G	A	C	A
	69%	79%	61%	56%	54%	54%

These rules are only approximately correct. The presence of promoters allow a very high transcription rate. Exercise: How to assign a score to the promoter.

This is the same as log likelihood ratio score, but this is often used concept, called **positional specific weight matrix** (PSWM). For example,

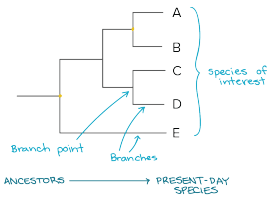
	1	2	3	4	5	6
A	0.1	0.76	0.1	0.61	0.56	0.1
C	0.1	0.1	0.1	0.1	0.1	0.04
G	0.03	0.1	0.2	0.1	0.2	0.04
T	0.77	0.04	0.6	0.19	0.14	0.82

We have $P_i(a)$ denoting the probability of having the letter a at position i. We define $q(a) = \frac{1}{4}$ to be random probability. Then

$$\begin{aligned} score(S_1 \cdots S_6) &= \log \frac{\Pr(S_1 \cdots S_6 \mid \text{promoter})}{\Pr(S_1 \cdots S_6 \mid \text{random})} \\ &= \log \prod_{i=1}^{6} \frac{P_i(S_i)}{q(S_i)} \\ &= \sum_{i=1}^{6} \log \frac{P_i(S_i)}{q(S_i)} \end{aligned}$$

In summary, HMM is a general model to predict some hidden states by examining emitted symbols. HMM can be used in gene prediction to harvest the codon bias and adjacent codon correlation. Gene prediction can use more information about the gene structure than codon bias. We only talked about prokaryote gene prediction. Eukaryote gene prediction is harder because of introns.

[3]在遗传学中，启动子（promoter）是指一段能使特定基因进行转录的DNA序列。启动子可以被RNA聚合酶辨认，并开始转录合成RNA。在RNA合成中，启动子可以和调控基因转录的转录因子产生相互作用，控制基因表达（转录）的起始时间和表达的程度，包含核心启动子区域和调控区域，就像“开关”，决定基因的活动，继而控制细胞开始生产哪一种蛋白质。



Part IV:

Evolutionary Tree Algorithms

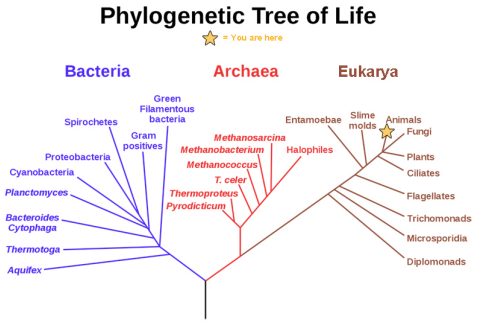
https://www.khanacademy.org/science/high-school-biology/hs-evolution/hs-phylogeny/a/phylogenetic-trees

15

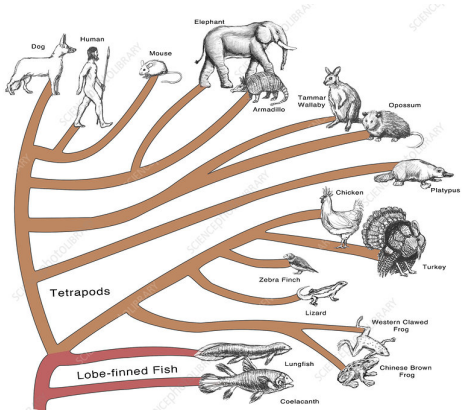
Phylogeny

15.1 Introduction & Chain Letter

A phylogeny[1] (also known as a phylogenetic tree): a diagrammatic hypothesis about the history of the evolutionary relationships of a group of organisms.



Above is one example: Phylogenetic Tree of Life.[2] Within the animal, we can see further details: Phylogenetic Tree of Animals[3]

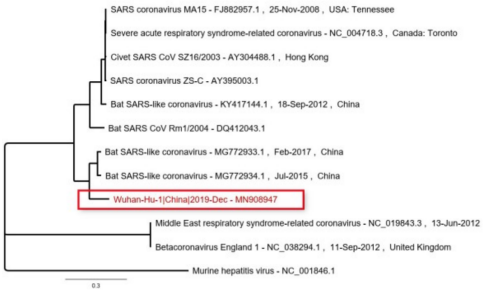


Phylogeny Tree of Coronavirus:

[1] 系统发生学（又称种系发生学、谱系发生学，简称谱系学，是研究及推理生物个体或群体（例如物种或种群）间演化史及关系的科学方法，属于系统分类学的一部分。

[2] https://courses.lumenlearning.com/wmopen-nmbiology1/chapter/phylogenetic-trees/

[3] https://www.sciencephoto.com/media/780514/view/evolutionary-tree-animals



In https://nextstrain.org/ncov/gisaid/global, we can see Genomic epidemiology of SARS-CoV-2 with global subsampling.

There are other examples for phylogenetic analysis. For example, outside biology: Study how the news articles copy each other. Study how chain letters evolve (our example to use). There are several terminologies:

- *Organisms*: the studied subjects.
- *Taxon* (taxa): A **taxon**[4] is a group of one (or more) populations of organism(s).
- *Species*: (usually) the largest group of organisms in which any two individuals of appropriate sexes can produce fertile offspring.

In 1997, on a Hong Kong mountain, the story begins. Charles H. Bennett (physicist at IBM) and Ming Li (professor at University of Waterloo) were hiking on Lion Rock. Charles Bennett collected 33 copies of chain letters[5] that were apparently from the same origin during 1980—1997. How does it work? Make 20 copies and send to your friends. Yes, then some good things will happen. No, no some bad things will happen. See https://en.wikipedia.org/wiki/Chain_letter for some examples. People make errors or typos when handwriting. This is like the evolution of the virus, with mutations. Here is an article: https://cs.uwaterloo.ca/~binma/cs482/chain-letter.pdf.

Why bother with chain letters? http://www.silcom.com/~barnowl/chain-letter/evolution.html Like a virus, it has reached billions of people, literally. Like a gene, they are about 2000 characters; It even resembles some subtle phenomenon in biological evolution! Like co-evolution in protein.

Here is an overall methods for constructing phylogeny:

- Character Based Method: Parsimony Method: (Find a tree topology so that the total number of mutations on the edges is the smallest.) Perfect Phylogeny, Maximum Likelihood.
- Distance Based Method: UPGMA, Neighbour Joining
- Quartet Method
- Whole Genome Phylogeny.

15.2 Character Based Method

15.2.1 Parsimony Method

The first category of phylogeny methods do three things:

- Define characters/features for each taxon.

[4] 分类单元又称分类单位或分类群，也可简称作类元或类群，是生物分类学上的基本概念，指一群拥有共同特征的生物的集合体，为分类学工作中的客观操作单位

[5] A chain letter is a message that attempts to convince the recipient to make a number of copies and pass them on to a certain number of recipients.

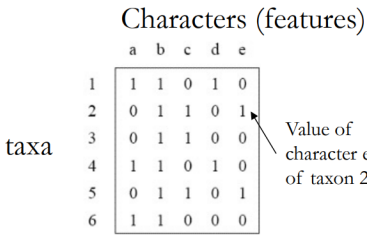
- Define a score function for each tree based on the characters.
- Find the optimal tree

A **character** is a "feature" in the species.

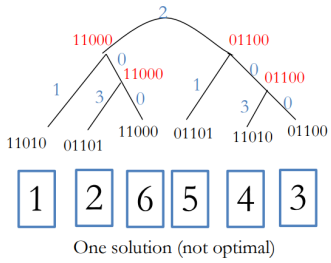
- Phenotype (表型): Vertebrate (脊椎动物) / invertebrate, Has hooves (蹄) / does not.
- Genotype (基因型): A letter in multiple sequence alignment
- Or in chain letter: The title is "Trust in lord . . . " or "With love all things are possible".

An **evolutionary tree** is a rooted and leaf-labeled binary tree.

For this method, *input* is an $n \times m$ matrix of aligned characters. (n taxa, m characters). In the case of a multiple alignment, these are columns with no gaps.



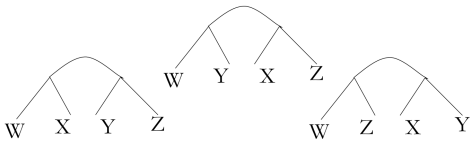
Output: A labeled tree with least number of mutations. Then the cost of tree will be the sum of all edge costs.



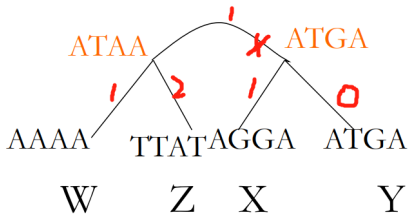
Consider an simple parsimony example. Suppose we have four taxa

- `W: AAAA`
- `X: AGGA`
- `Y: ATGA`
- `Z: TTAT`

There're only 3 possible (unrooted) trees on 4 taxa.



Which has the least mutations?



In this case, we need 5 mutations. The other 2 require 6. So the "cheapest" tree joins W and Z on one side and X and Y on the other.

Where is the root of the tree? We said that phylogeny is a rooted tree, and this example doesn't have a root yet. The root doesn't matter because the distance is symmetric. For example, root can be at $\times$. No matter where we put the root, it doesn't change the cost of the tree. For simplicity, we can just copy one of the children leaf.

In that problem, we avoided the **ancestor reconstruction**. We need to label the internal nodes to calculate the cost. For a given topology, how to construct the ancestors? (in order to calculate the score of the tree)

The input includes the given topology and the labels of on all leaves. The goal is to label the internals to minimize the cost.

First observation: We can solve each column (character) separately. So we can just solve for 1-character strings. Algorithm by Sankoff: tree-based dynamic programming.

For every node u of the tree and letter a of the alphabet Σ, let $D[u, a] =$ min number of mutations in T_u if u's label is a, where T_u is the subtree rooted at u.

Let r be the root. We want $min_x D[r, x]$. For a leaf node v, if the character at leaf v is a, then $D[v, a] = 0$, and $D[v, b] = 1$ for all other letters b. For an internal node u, with children v and w, suppose we know all of the values of $D[v, *]$ and $D[w, *]$. How to compute $D[u, *]$?

If we put letter "a" at node u, the cost of the left branch of the tree is the minimum of

$$\begin{cases} D[v, a], & \text{if } v = a \\ 1 + \min_{b \neq a} D[v, b], & \text{if } v \neq a \end{cases}$$

The same argument holds for the right branch. So

$$D[u, a] = \min(D[v, a], 1 + \min_{b \neq a} D[v, b]) + \min(D[u, a], 1 + \min_{b \neq a} D[w, b])$$

The order of computation is depth-first. For time complexity, we need enumerate for all nodes u and for all letter a, and within the recurrence relation, we also need to try every single letter. Thus $O(n \times \sigma^2)$ where $\sigma := |\Sigma|$.

We notice that

$$\min(D[v, a], 1 + \min_{b \neq a} D[v, b]) \equiv \min(D[v, a], 1 + \min_{b \in \Sigma} D[v, b])$$

and $\min_{b \in \Sigma} D[v, b]$ does not depend on a. Thus we can take it outside the inner for loop:

- for every u:
- $x := \min_{b \in \Sigma} D[v, b]$.
- for every a:
- apply the recurrence relation: $D[u, a] = \min(D[v, a], 1 + x) + \cdots$.

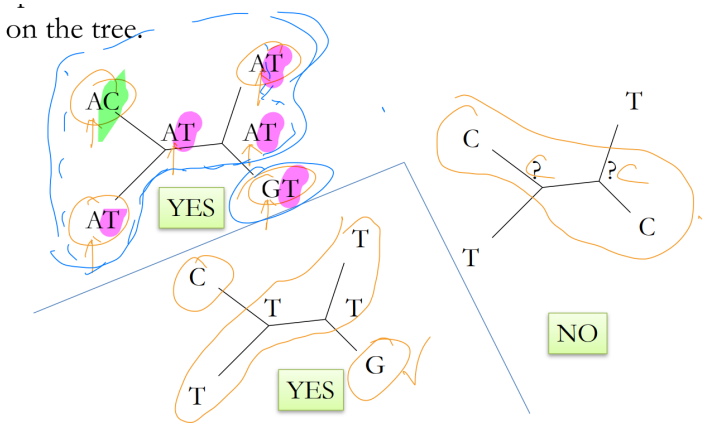
So $\min_{b\in\Sigma} D[v,b]$ is only computed once, not once every letter a for $\min_{b\neq a} D[v,b]$. If the tree is binary, this algorithm takes $O(n\sigma)$ time, since it's just $O(\sigma)$ time at each node.

Now we can use dynamic programming to compute the cost of tree. **Parsimony Method**: Go through all tree topologies, find a tree topology so that the total number of mutations on the edges is the smallest. This problem is NP-hard. Algorithm: For each possible tree topology, uses DP to compute cost. Output the best tree.

Suppose there are $f(n)$ trees on n taxa. Total runtime is $O(nm\sigma f(n))$ where $f(n) = 1\cdot 3\cdot\cdots\cdot(2n-5) \approx n!2^n$.

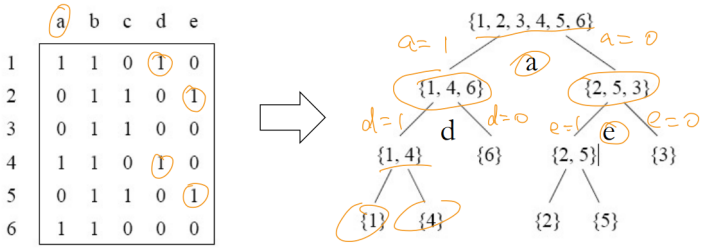
15.2.2 Perfect Phylogeny

Basically, perfect phylogeny is best we can achieve via parsimony method. A **perfect phylogeny** is such that for every character (every column), all species with the same state of that character is a connected component on the tree.



For the "no" example, no matter how we put in "?", either C or T, no way to have connected components for T and C at the same time.

Parsimony method is slow. If there is a perfect phylogeny, here is an algorithm to find it efficiently. Start with a set of all taxa. Find a character and split the set into two. Recursion until each set has only one taxon.



So in each node, we keep splitting. For example, for the root node, we split either $a = 1$ or $a = 0$ to get two children nodes.

For binary characters, this algorithm is a polynomial time algorithm. If there is a perfect phylogeny, it outputs the perfect phylogeny. Equivalently, if the output is not a perfect phylogeny, then there is no perfect phylogeny for the input.

Theorem

If there is a perfect phylogeny for the input, and there are constant number of states for the characters, then a perfect phylogeny can be computed in polynomial time.

r states, n taxa, m characters: $O(2^{2r}nm^2)$.

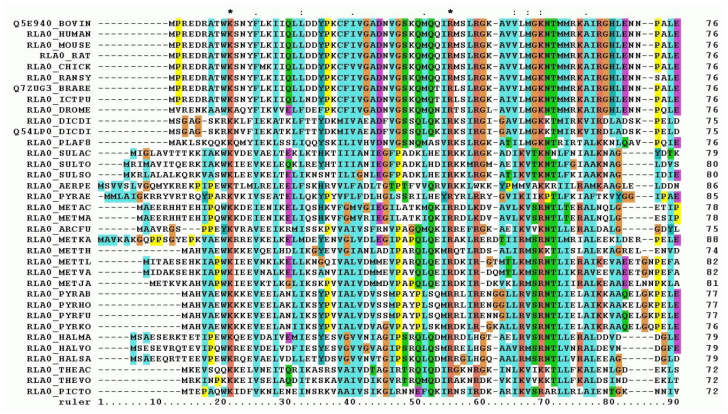
If a column has k different states, then any phylogeny requires at least $k-1$ mutations for the column. A perfect phylogeny only has $k-1$ mutations for the column. Thus, perfect phylogeny is the best you can get for parsimony. But not all input matrix can cause a perfect phylogeny.

15.2.3 Maximum Likelihood

It's same idea as parsimony, but makes score function better. The score function used in parsimony (and perfect phylogeny) is too simple, especially when sequencing data become available. For example: the multiple alignment of proteins can be used as the input:

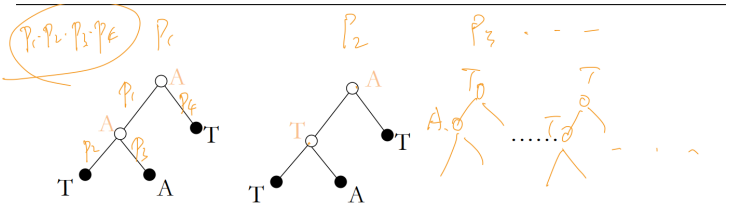
- Thousands of columns (characters).
- Mutations between different pairs of amino acids have different rates.
- Different substitution matrices on different columns.

The maximum likelihood method aims to provide a better scoring function.



Here is a multiple sequence alignment. We can see some columns are more important than other columns. Max likelihood method starts with a multiple alignment. Different columns may have different substitution frequency matrix.

How do we define the score? Remember, each column is independent, so we can only focus on one column.



- For each possible tree topology T, (e.g., a tree structure like above) for each possible internal node assignment, and calculate the probability based on the substitution matrix.

  For example, in the first labeling, we have labeled the edge with p_1, p_2, p_3, p_4, then this assignment gives $P_1 = p_1p_2p_3p_4$.

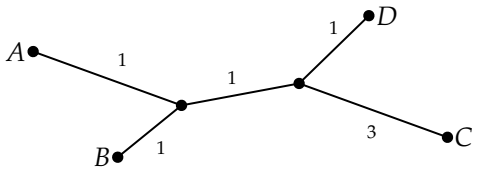
- For each tree T, add up all probabilities of all possible internal nodes, $P_1 + P_2 + P_3 + P_4 + \cdots$. This is the likelihood of the input tree T. Figure shows a single column of the multiple alignment.
- Find the tree topology that maximizes the likelihood.

Often more accurate than other methods. Very time consuming because we have to enumerate all possibilities. Usually heuristic algorithms and dynamic programming algorithms are used to assist the search and estimation of the likelihood. If desired, one can also allow the change of the edge length (the mutation rate at each edge).

Software available: e.g. PhyML, can help compute the max likelihood tree.

15.3 Distance Based Method

Input of distance based methods is an $n \times n$ distance matrix $d(i, j)$. For example, in the ideal situation, given a true evolution history, and the length of mutation, we then can fill in distance matrix.



	A	B	C	D
A	0	2	6	4
B	2	0	6	4
C	6	6	0	4
D	4	4	4	0

We want to compute a tree with n leaves, with edge weights. $T(i, j)$ is the distance of two leaves on the tree. We want to minimize $\sum_{i,j} |d(i, j) - T(i, j)|^2$, where $d(i, j)$ is our input, and $T(i, j)$ is the distance we construct with our algorithm. This is also NP-hard. So we use heuristics.

15.3.1 UPGMA

Unweighted Pair Group Method with Arithmetic mean.

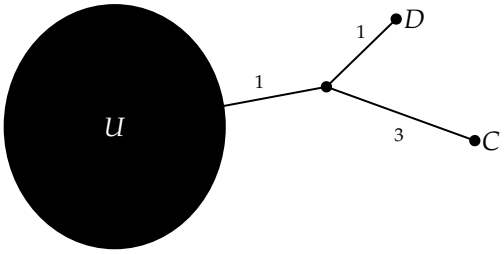
We focus on a species, **siblings**, which are two leaves connected to the same parent. Can we find the siblings in the distant matrix? If we can confidently determine which are siblings, we can merge them together as a supernode. We keep doing this until we get only one node. At the end, we can reverse this process to expand this tree.

It is a heuristic method with no performance guarantee. At each time, it finds i, j with the minimum distance. Merge the two taxa i, j into a new one u. Update the distance matrix. For any k, let $d(u, k) = (d(i, k) + d(j, k))/2$. Then recurse.

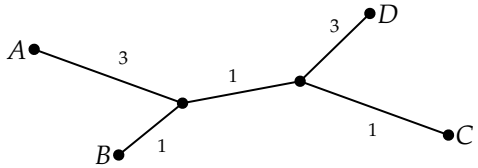
Now let's try to apply this idea to the previous example. We notice that A, B is the closest, we then merge them to U and get a new matrix:

	U	C	D
U	0	6	4
C	6	0	4
D	4	4	0

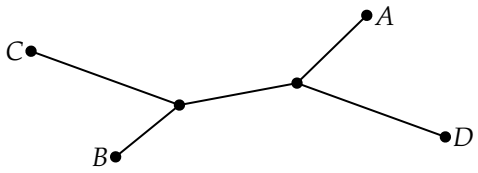
and its corresponding graph:



There's a big problem with UPGMA method while it's still being used widely. Suppose we are given such a real phylogeny tree:



We notice that the shortest path between any two pairs is BC, of length 3. Then if we run UPGMA method on the distance matrix produced by this tree, we will merge B and C, then we will end up with the tree below:



which is the wrong tree.

15.3.2 Neighbor Joining

Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol. 1987 Jul;4(4):406-25.

Neighbor Joining uses a similar idea as UPGMA. But it uses a more sophisticated formula to determine the two neighbors to be joined.

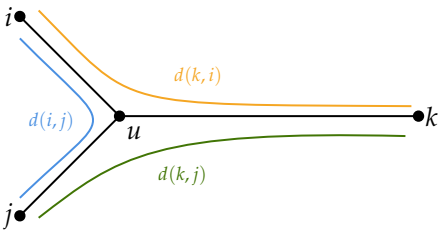
$$Q(i,j) = (r-2)d(i,j) - \sum_{k=1}^{r} d(i,k) - \sum_{k=1}^{r} d(j,k) \tag{15.1}$$

- Find the minimum $Q(i,j)$, merge i, j to a new node u.
- Update $d(k,u) = (d(k,i) + d(k,j) - d(i,j))/2$
- Recursion on the remaining $r-1$ nodes.

With some rearrangements of (15.1), we notice that

$$-Q(i,j) = \left(\sum_{k=1}^{r} (d(i,k) + d(j,k) - d(i,j))\right) + 2d(i,j) \tag{15.2}$$

We can visualize the update procedure as follows:



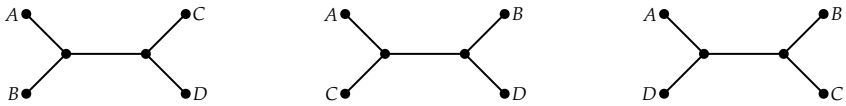
So the update formula computes the distance between another species to the parent of i and j.

Therefore, from (15.2) we see that to minimize $Q(i, j)$, we want to maximize RHS of (15.2), especially the summation in RHS.

This method fixes the previous issue: $Q(B, C) > Q(A, B)$, which correctly picks up the siblings. In the paper, they proved that if the error in the input matrix is no greater than the minimum edge weight of the tree, then the neighbor joining method will give us the correct tree all the time.

15.4 Quartet Methods

It's hard to construct a big tree when we have many species, because it's hard to enumerate all of them. Instead, for each group of four species, construct a tree of 4 (quartet), using your most favorite method, say maximum likelihood.



Then find a tree that is most consistent with all the quartets. The problem is NP-hard (to find the tree with least error). There is a PTAS to do this. [6]

There are other proposals for constructing the phylogeny tree, but they never get popolar.

15.5 Challenges in Phylogeny of Chain Letter

Parsimony or maximum likelihood: How do we know what is a "character" - a feature to look at?

- In case of a chain letters? It's hard to come up with all the characters that can be used for the phylogeny. Maximum likelihood is also hard because of the mutation matrix.
- In case of **whole genome phylogeny**? We have n species and all the genome sequence completely, so we want to build the phylogeny tree using all the information in the genome. Then it's hard to do multiple sequence alignment, e.g., genome rearrangement event.

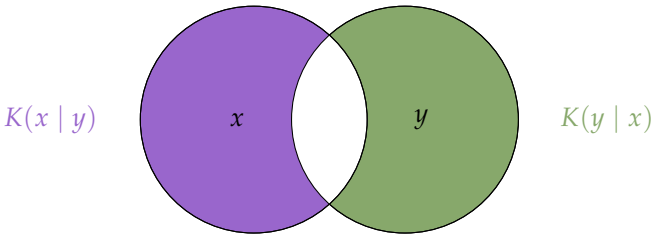
or use distance based method? Which distance to use? Edit distance doesn't work due to genome rearrangement event.

There's a "universal" solution: **information distance**.

We need a notion of **Kolmogorov Complexity**: the length of a shortest computer program (in a predetermined programming language) that produces the object as output (wiki).

$K(x)$ is the shortest program length to output x.

$K(y \mid x)$ is the length of shortest program to take x as input and output y. If x is equal to y, then the program is just doing copy. So $K(y \mid x)$ actually reflects the *additional* information y has, to x.



[6] (T. Jiang, P. Kearney, and M. Li. Orchestrating quartets: approximation and data (FOCS'98))

Then the information distance between x and y is defined as

$$d(x,y) = \frac{\max\Big(K(x \mid y), K(y \mid x)\Big)}{\max\Big(K(x), K(y)\Big)}$$

Kolmogorov complexity is incomputable, but we can use compression in practice.

Now we can reconstruct history of chain letters. We estimate $K(x \mid y) \approx K(xy) - K(y)$ where xy denotes the concatenation. Similarly for $K(y \mid x)$.

For each pair of chain letters (x, y) we computed $d(x, y)$, hence a distance matrix. A DNA compression program (expand the alphabet size to 26) is used to compute the information distance. Using Neighbor Joining to construct their evolutionary history based on the $d(x, y)$ distance matrix. The resulting tree is an almost perfect phylogeny: distinct features are all grouped together. We then see the result is good.

16

Suffix Tree and Array

So far we learned how to find "approximate" matches - the alignments. And they are difficult. Finding exact matches are much easier. Suffix Tree and Array are helpful for **string matching** in bioinformatics.

String matching is to search for a short string P of length m in a large text T of length n.

There are some applications: Keyword searching, DNA reads mapping. Keyword searching has application in bioinformatics, for example, next-generation sequencing.

> Next-generation sequencing (NGS) is a massively parallel sequencing technology that offers ultra-high throughput, scalability, and speed. The technology is used to determine the order of nucleotides in entire genomes or targeted regions of DNA or RNA.

Src: https://www.illumina.com/science/technology/next-generation-sequencing.html.

In short, we are given a reference genome, and we want to map DNA reads back to the reference genome.

There are two scenarios for string matching:

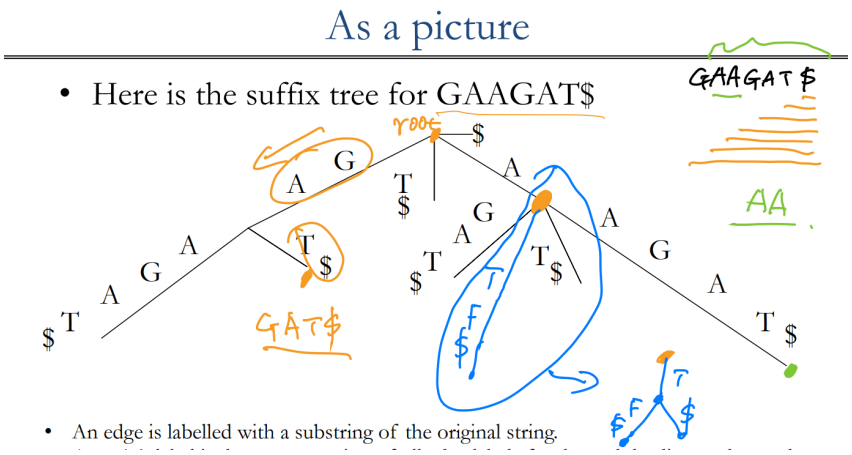
- **Type I: Match only once**. E.g. KMP algorithm and Apostolico-Giancarlo algorithm. $O(m)$ to preprocess, and $O(n)$ to match.
- **Type II: Match multiple patterns multiple times**. Better index T first to speed up the matching time.

This actually requires one of the data structures: suffix tree or suffix array. They are related to each other, and one can show that these two are equivalent. Suffix tree and array are two data structures for this purpose.

History skipped.

16.1 Suffix Tree

We can construct suffix tree for `GAAGAT$`. We typically put $ to indicate the end of string.



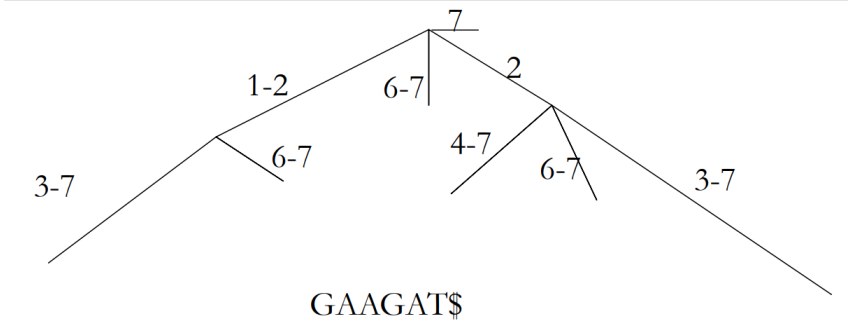
Several properties:

- An edge is labelled with a substring of the original string.
- A node's label is the concatenation of all edge labels for the path leading to that node.
- The path from the root, r, to any leaf x is a suffix of the string S.
- Suppose there is a special "end-of-string" character, each suffix will end at the leaf.
- Each internal node has at least 2 children.
- Edge labels to the child nodes of an internal node start with different letters. (shown in blue: we can "merge")

16.1.1 Application I. Search for a substring

If we have suffix tree, we can search for a substring. We note that *any substring of S is a prefix of a suffix*.

Is the string x a substring of S? Start at the root, and follow paths labelled by the characters of x. If you can get to the end of x, then yes, it is.

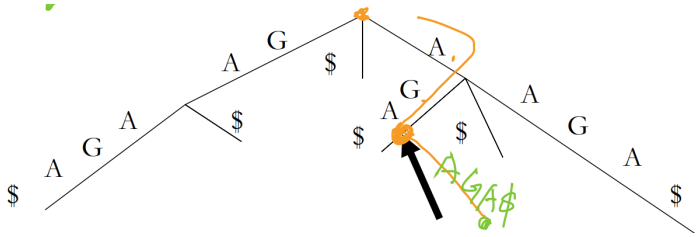


Each edge doesn't need to be labelled with a string, but just with starting and ending in the sequence. This is the same suffix tree as before, but in linear space.

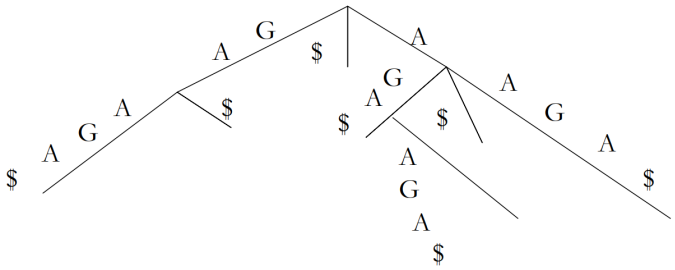
How to construct a suffix tree? There is a linear time algorithm to construct a suffix tree. (We will not study it.)

We'll examine a quadratic-time algorithm (quite intuitive). The idea is to start with an empty tree, and iteratively add more suffixes into the tree (from shortest to longest).

Suppose the following is the suffix tree for `GAAGA$`, add another suffix `AGAAGA$`.



First, follow the edges for `A` and for `GA` from the root. Then split after the `A` since the only path in the tree is for `$`, and we have an `A`, instead. Add a new edge for `AGA$`. This yields this new tree:

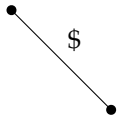


Algorithm 15: Suffix tree quadratic time construction

Input: A string S of length m over a finite alphabet. The last character of S is a unique `$` character.

1 We'll build the suffix tree from right to left: $S[m..m], S[m-1..m], S[m-2..m], \ldots$

2 Begin with this tree:



3

4 **for** $i \leftarrow m..1$ **do**

5 Follow the letters of $S[i..m]$ along the edges of the tree T.

6 When we reach a point where no path exists, break the current edge and add a new edge for what is left.

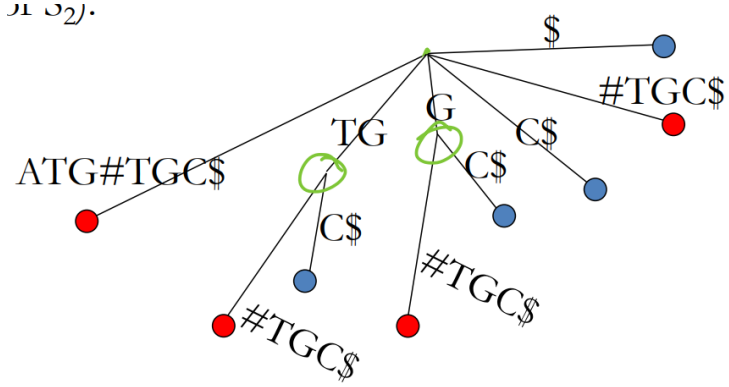
Time complexity: $O(m^2)$. (Remember: The best algorithm has linear time.)

16.1.2 Application II: Longest Common Substring

What's the longest substring common to both S_1 and S_2? Straightforward algorithm will try to compare all substrings of equal length. This takes cubic time. Can we do better?

We can do local sequence alignment, which is quadratic time. But the best algorithm is to use suffix tree, which is in linear time.

We build a suffix tree for $S = S_1\#S_2\$$, where # and $ are unique characters. All suffixes of S_1 end with an edge including $\#S_2\$$. So we can label whether a leaf belongs to S_1 or S_2. In the example below, we have $ATG\#TGC\$$.



Substrings are prefixes of suffixes, i.e. internal and leaf nodes of the tree. Each common substring is the prefix of at least two suffixes, each from an input string (S_1 or S_2). So we are looking for *internal node*, with mixed color leaves, then nodes satisfying these two properties are common substring. We go through all possible such nodes to find the longest common substring. Circled ones in the picture are two such nodes.

Algorithm 16: Longest common substring with suffix tree

1. Build suffix tree of $\#S_2\$$.
2. Color all leaf nodes: red if v's label is a substring of S_1, blue if it's a substring of S_2.
3. Color all internal nodes from bottom up: red (or blue) if all child nodes are red (or blue), purple if otherwise.
4. Find the purple node with longest path label.

Linear time, linear space.

Sketch proof of correctness Let t be the longest common substring. Follow the path label t starting from the root. The path can't stop in the middle of the edge - otherwise t is not the longest. Then the path has to stop at an internal node. And it has to be purple.

16.1.3 Application III: Maximal Unique Match

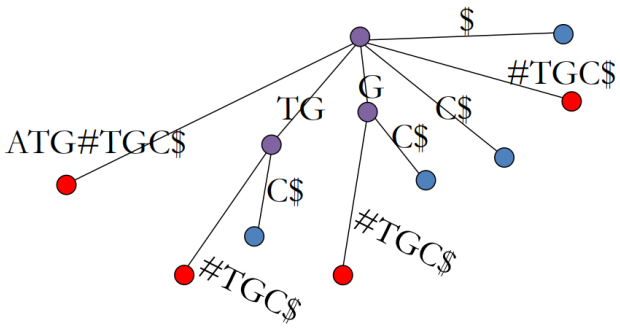
Given two strings, a **MUM (Maximal Unique Match)** is a string that occurs exactly once in each string, and is maximal (can't be extended either way and still be a match).

For example, `ATGAATC` vs. `AGATC`

- `AT` is not, because not unique.
- `G` is not, not maximal because `GA` is longer.
- `GA` is a MUM.
- `ATC` is a MUM.

We build a suffix tree for $S = S_1\#S_2\$$, and color the nodes as in the longest common substring algorithm.

Example:
ATG#TGC$



Each MUM must be a purple internal node that has *exactly two leaf children*: one red and one blue.

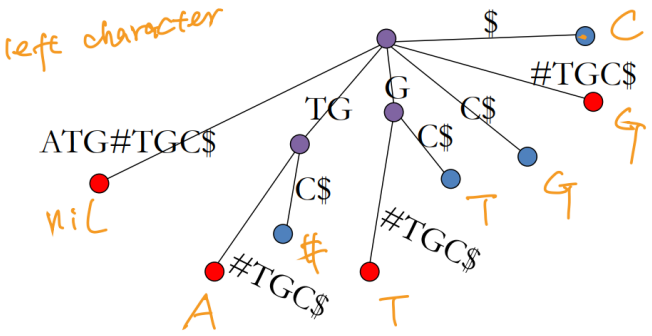
- Having exactly two leaf children ensures the uniqueness.
- We also want maximality:
  - it is shared by the two strings.
  - it can't extend to the right by an additional letter and still be shared.

But a purple internal node may not be a MUM: only because the two occurrences may still extend to the left. In the example,

- Node G is not: For G's two occurrences, the left character are both T.
- Node TG is: For TG's two occurrences, the left characters are A and #, respectively.

But it is easy to compute the left character of each leaf: it is a suffix, and we know its path's starting position in the original string. We put the left character of each leaf:

Example:
ATG#TGC$



Algorithm 17: Maximal unique match with suffix tree

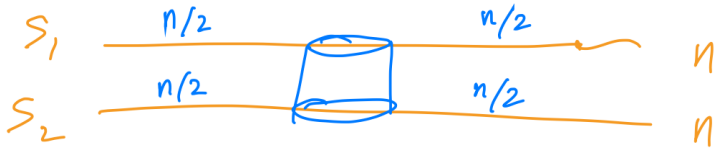
```
1 Build suffix tree of #S_2$.
2 foreach leaf v do
3     left(v) := letter at left of suffix v.
4 Find the internal nodes that
5 • Have exactly two child leaves
6 • The two child leaves are two suffixes from S_1 and from S_2, respectively.
7 • The two child leaves must have two different left characters.
```

This is linear time. After find all MUMs, use them as anchor to speed up global alignment.

16.1.4 MUMMER: Large-scale Global Alignment

The idea of MUM has application in bioinformatics. For example, large-scale global alignment. Previously, given two genomes of length n, cost is $O(n^2)$. The idea is

- Pick some "anchors" through which the true alignment is very likely to fall.
- This breaks up into two separate alignments.
- Align the regions between the anchors either recursively or just using classical global alignment tools.



Now the cost is $O(2 \times (n/2)^2) = O(n^2/2)$. If we break into k parts, then it's $O(n^2/k)$.

MUMs are good anchors: maximal, unique, match. First program that does so: MUMMER by Delcher et al.

16.2 Suffix Array

Suffix tree is not a compact data structure as it has a lot of pointers. A **suffix array** stores the positions in a string. Each position is an integer so this is a length n integer array. Each position corresponds to a suffix starting at this position. The suffix array is sorted according to the string order of the corresponding suffixes.

Consider the string `AGAAGAT`. We have the starting positions for suffixes:

$$1 = \texttt{AGAAGAT}$$
$$2 = \texttt{GAAGAT}$$
$$3 = \texttt{AAGAT}$$
$$4 = \texttt{AGAT}$$
$$5 = \texttt{GAT}$$
$$6 = \texttt{AT}$$
$$7 = \texttt{T}$$

Then we sort according to the alphabetic order on the right, we have

$$3 = \texttt{AAGAT}$$
$$1 = \texttt{AGAAGAT}$$
$$4 = \texttt{AGAT}$$
$$6 = \texttt{AT}$$
$$2 = \texttt{GAAGAT}$$
$$5 = \texttt{GAT}$$
$$7 = \texttt{T}$$

This gives us $3, 1, 4, 6, 2, 5, 7$.

With sorted suffixes ($O(n \log n)$ trivially, but we can do linear time with the algorithm in next section), we can do query quickly with binary search. Suppose the query substring is of length m:

- $O(m \log n)$ if implemented straightforwardly
- $O(m + \log n)$ if with an auxiliary data structure called longest common prefix (LCP) array. We do not study this but we should be aware of this fact.

16.3 Skew Algorithm For Suffix Sorting

The idea is similar to merge sort, but we can't afford the cost of recursing on both smaller parts.

Let $S_0, S_1, \ldots, S_{n-1}$ be all the n suffixes. S_i starts at i-th position. Skew algorithm uses divide and conquer. But it divides the problem into unequally sized parts. We have two sets:

$$\mathrm{SA}^0 = \{S_i : i = 0 \bmod 3\} \qquad \mathrm{SA}^{12} = \{S_i : i \neq 0 \bmod 3\}$$

General idea We assume SA^{12} is sorted already, and learn the other two steps first.

1. Sort SA^{12} recursively. (pad with 0, 3-mer, and renaming)
2. Sort SA^0 in linear time. (radix sort)
3. Merge sorted SA^0 and SA^{12} in linear time. We compare first one or two letters, and the remaining shorter substrings.

Other sources on skew algorithm:

- https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/suffixarrays.pdf
- http://www.cs.cmu.edu/~guyb/paralg/papers/KarkkainenSanders03.pdf
- https://gist.github.com/markormesher/59b990fba09972b4737e7ed66912e044 (recommended)