Lecture 2: Pairwise Alignment
Main source

![Book Cover](image.jpg)

**ALGORITHMS ON STRINGS, TREES, AND SEQUENCES**

*Computer Science and Computational Biology*

**Dan Gusfield**

![Author Photo](image2.jpg)
Why compare sequences?

Human hexosaminidase A vs Mouse hexosaminidase A
Sequence Alignment

• **The problem**: Comparing two sequences while allowing certain mismatches between them.

• **Main motivation**:
  - Comparing DNA seqs and proteins from databases,
    • Comparing two or more sequences for similarity
    • Searching databases for related sequences and subsequences
    • Finding informative elements in protein and DNA sequences
Alignment definition

- **Input**: Two sequences of possibly different lengths
- **Goal**: Space the sequences so that they have the same length and no two spaces are matched.

\[
\begin{array}{c}
acbd \\
cadb \\
\end{array}
\rightarrow
\begin{array}{c}
ac- - \\
-cadb - \\
\end{array}
\]

\[
\begin{array}{c}
acbc \\
acdc \\
\end{array}
\rightarrow
\begin{array}{c}
acbc \\
acdc \\
\end{array}
\]

\[
\begin{array}{c}
acbd \\
cadb \\
\end{array}
\rightarrow
\begin{array}{c}
acbd \\
acdbb \\
\end{array}
\]
Alignment scoring: Similarity vs. Difference

- **Resemblance** of DNA sequences of different organisms explained by common ancestral origin
- **Differences** are explained by mutation:
  - Insertion
  - Deletion
  - Substitution
- **Distance** between two sequences is the minimum (weighted) sum of mutations transforming one into the other.
- **Similarity** of two sequences is the maximum (weighted) sum of resemblances between them.
Nomenclature

- **Computer Science:**
  - String, word
  - Substring (contiguous)
  - Subsequence
  - Exact matching
  - Inexact matching

- **Biology:**
  - Sequence
  - Subsequence
  - N/a
  - Alignment

We shall use the **biology** nomenclature
Simplest model: Edit Distance

The *edit distance* between two sequences is the min no. of edit operations (single letter insertion, deletion and substitution) needed to transform one sequence into the other.

ACCTGA and AGCTTTA

\[
\begin{align*}
\text{ACCTGA} & \quad \text{AGCTTTA} \\
\text{ACCTGA} & \quad \text{A\_CCTGA} \\
\text{AGCCTGA} & \quad \text{AGCTT\_A} \\
\text{AGCTTTGA} & \quad \text{ACCTTGA} \\
\text{AGCTTTA} & \quad \text{AGCTTTA}
\end{align*}
\]

3 operations

2 operations

**dist=2**
Alignment

24 matches,
Subs: TA, AG, GC, CG
Indels -T, G-, G-, A-, T-
Distance I: match 0, subs 1, indel 2 \( \Rightarrow \) dist = 14
Alignment

SEQ 1     GTAGTACAGCT−CAGTTGGGATCACAGGCTTCT
          ||||  ||   ||||   ||||   ||||   ||||
SEQ 2     GTAGAAGGCGCTTCAGTTG---TCACAGCGGTTC−

• 24 matches, Subs: TA, AG, GC, CG, Indels -T, G-, G-, A-, T-

• Distance II: match 0, d(A,T)=d(G,C)=1, d(A,G)=1.5 indel 2
  ➔ dist=14.5

• Similarity I: match 1, subs 0, indel -1.5
  ➔ similarity =16.5

• General setup: substitution matrix $S(i,j)$, indel $S(i,-)$

  Usually symmetric, Alignment - to - not allowed.
Models for Alignment

**Problem**: Global Alignment

**Input**: Two sequences $S$, $T$ of roughly the same length

**Goal**: Find an optimal alignment between them

**Problem**: Local Alignment

**Input**: Two sequences $S$, $T$

**Goal**: Find a subsequence of $S$ and a subsequence of $T$ with optimal alignment (most similar subsequences).
Models for Alignment (2)

**Problem:** Ends free alignment (End space-free alignment)

**Input:** Two sequences S, T

**Question:** Find an optimal alignment between subsequences of S and T where at least one of these subsequences is a prefix of the original sequence and one (not necessarily the other) is a suffix.

*gap*: max contiguous run of spaces in a single sequence within a given alignment

**Problem:** Alignment with Gap Penalty

**Input:** Two sequences S, T

**Question:** Find an optimal alignment between them, given a gap penalty function.

measures the cost of a gap as a (nonlinear) function of its length
Global Alignment Problem:

**Input**: Two sequences $S = s_1...s_n$, $T = t_1,...,t_m$ ($n \sim m$)

**Goal**: Find an optimal (max. similarity) alignment under a given scoring function.
How many alignments are there?

• Each alignment matches $0 \leq k \leq \min(n,m)$ pairs.

• \#alignments with $k$ matched pairs is $\binom{n}{k}\binom{m}{k}$

\[
N = \sum_{k=0}^{\min(n,m)} \binom{n}{k}\binom{m}{k} = \binom{n+m}{\min(n,m)}
\]
Global Alignment Algorithm

• First dynamic programming solution by Needleman & Wunsch (70); improved later by Sankoff (72).

Notation:
• $\sigma(a,b)$: the score (weight) of the alignment of character $a$ with character $b$.
• $V(i,j)$: the optimal score of the alignment of $S'=s_1...s_i$ and $T'=t_1...t_j$ ($0 \leq i \leq n$, $0 \leq j \leq m$)
**Lemma**: \( V(i,j) \) has the following properties:

- **Base conditions**:
  - \( V(i,0) = \sum_{k=0}^{i} \sigma(s_k,-) \)
  - \( V(0,j) = \sum_{k=0}^{j} \sigma(-,t_k) \)

- **Recurrence relation**:
  - \( \forall 1 \leq i \leq n, 1 \leq j \leq m: V(i,j) = \max \begin{cases} V(i-1,j-1) + \sigma(s_i,t_j) \\ V(i-1,j) + \sigma(s_i,-) \\ V(i,j-1) + \sigma(-,t_j) \end{cases} \)

Alignment with 0 elements \( \equiv \) spaces.

\( S' = s_1...s_i \) with \( T' = t_1...t_j \) and \('-'\) with \( t_j \).
Optimal Alignment - Tabular Computation

- Use dynamic programming to compute $V(i,j)$ for all possible $i,j$ values:

```plaintext
for i=1 to n do
begin
  For j=1 to m do
  begin
    Calculate $V(i,j)$ using $V(i-1,j-1), V(i,j-1), V(i-1,j)$
  end
end
```

Snapshot of computing the table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Costs: match 2, mismatch/indel -1
Optimal Alignment - Tabular Computation

• Add back pointer(s) from cell \((i,j)\) to father cell(s) realising \(V(i,j)\).

• Trace back the pointers from \((m,n)\) to \((0,0)\).

Backtracking the alignment
### Example

**x = AGTA**  
**y = ATA**

**m = 1**  
**s = -1**  
**d = -1**

<table>
<thead>
<tr>
<th></th>
<th>i = 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>j = 0</td>
<td>`1`</td>
<td>``</td>
<td>A</td>
<td>G</td>
<td>T</td>
</tr>
<tr>
<td>Id</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**V(1, 1) = max{V(0,0) + s(A, A), V(0, 1) + d, V(1, 0) + d} =**

\[
\begin{align*}
\max\{0 + 1, -1 - 1, -1 - 1\} &= 1
\end{align*}
\]
+10 for match, -2 for mismatch, -5 for space
Traceback can yield both optimum alignments
Alignment Graph

\[(0,0)\]

\[\sigma(s_{i+1},-)\]

\[\sigma(-, t_{j+1})\]

\[\sigma(s_{i+1}, t_{j+1})\]
Alignment Graph

**Definition:** The alignment graph of sequences $S=s_1...s_n$ and $T=t_1...t_m$, is a directed graph $G=(V,E)$ on $(n+1)x(m+1)$ nodes, each labeled with a distinct pair $(i,j)$ ($0 \leq i \leq n$, $0 \leq j \leq m$), with the following weighted edges:

- $((i,j), (i+1,j))$ with weight $\sigma(s_{i+1}, -)$
- $((i,j), (i,j+1))$ with weight $\sigma(-, t_{j+1})$
- $((i,j), (i+1,j+1))$ with weight $\sigma(s_{i+1}, t_{j+1})$

**Note:** a path from node $(0,0)$ to node $(n,m)$ corresponds to an alignment and its total weight is the alignment score.

**Goal:** find an optimal path from node $(0,0)$ to node $(n,m)$
Complexity

- Time: $O(mn)$ (proportional to $|E|$)
- Space to find opt alignment: $O(mn)$ (proportional to $|V|$)
- Space is often the bottleneck!
- Space to find opt alignment value only: $O(m+n)$. Why?
- Can we improve space complexity for finding opt alignment?
Warm-up questions

How do we efficiently compute the optimal alignment scores of $S$ to each prefix $t_1 \ldots t_k$ of $T$?

How do we efficiently compute the optimal alignment score of the sequence suffixes $s_{i+1} \ldots s_n$ and $t_{j+1} \ldots t_m$?
Reducing Space Complexity

\[ V^*(n-i,m-j) = \text{opt alignment value of } s_{i+1}...s_n \text{ and } t_{j+1}...t_m \]

**Lemma:** \[ V(n, m) = \max_{0 \leq k \leq m} \left\{ V\left(\frac{n}{2}, k\right) + V^*\left(\frac{n}{2}, m-k\right) \right\} \]

**Pf:** \[ \max\{...\} \leq V(n,m): \]
- \( \forall \) position \( k' \) in \( T \), \( \exists \) alignment of \( S \) and \( T \) consisting of:
  - an opt alignment of \( s_{1}...s_{n/2} \) and \( t_{1}...t_{k'} \) and
  - a disjoint opt alignment of \( s_{n/2+1}...s_n \) and \( t_{k'+1}...t_{m} \).
• \( \max\{\ldots\} \geq V(n,m) : \)
  - For an optimal alignment of \( S \) and \( T \), let \( k' \) be the rightmost position in \( T \) that is aligned with a character at or before position \( n/2 \) in \( S \). Then the optimal alignment of \( S \) and \( T \) consists of:
    • an alignment of \( s_1 \ldots s_{n/2} \) and \( t_1 \ldots t_{k'} \) and
    • a disjoint alignment of \( s_{n/2 + 1} \ldots s_n \) and \( t_{k'+1} \ldots t_m \).
'Divide & Conquer' Alg (Hirschberg '75)

- Compute opt cost of all paths from start, to any point at centerline
- Compute opt cost of back paths from end to any pt at centerline
- Pick centerline pt with opt sum of the two costs
- Continue recursively on the subproblems
Linear-space Alignments

\[ mn + \frac{1}{2} mn + \frac{1}{4} mn + \frac{1}{8} mn + \frac{1}{16} mn + \ldots = 2 mn \]
Hirschberg Alg in more detail

$k^*$ - position $k$ maximizing $V(n/2,k) + V^*(n/2,m-k)$

Proved: $\exists$ opt path $L$ through $(n/2,k^*)$

Def: $L_{n/2}$ - subpath of $L$ that

- starts with the last node in $L$ in row $n/2-1$ and
- ends with the first node in $L$ in row $n/2+1$
$L_{n/2}$
Lemma: $k^*$ can be found in $O(mn)$ time and $O(m)$ space. $L_{n/2}$ can be found and stored in same bounds.

Run DP up to row $n/2$, getting values $V(n/2, i)$ for all $i$ and back pointers for row $n/2$.

Run DP backwards up to row $n/2$, getting values $V^*(n/2, i)$ for all $i$ and forward pointers for row $n/2$.

Compute $V(n/2,i)+V^*(n/2,m-i)$ for each $i$, get maximizing index $k^*$.

Use back pointers to compute subpath from $(n/2,k^*)$ to last node in row $n/2-1$.

Use forward pointers to compute subpath from $(n/2,k^*)$ to first node in row $n/2+1$.

$O(mn)$ time, $O(m)$ space

$O(m)$ time, space

$O(m)$ time, space incl storage
Full Alg and Analysis

- Assume time to fill a p by q DP matrix: \( cpq \)
- \( \Rightarrow \) time to compute rows \( V(n/2,..), V^*(n/2,..) \): \( cmn \)
- \( \Rightarrow \) time \( cmn \), space \( O(m) \) to find \( k^*, k_1, k_2, L_{n/2} \)
- Recursively solve top subproblem of size \( \leq nk^*/2 \), bottom subproblem of size \( \leq n(m-k^*)/2 \)
- Time for top level \( cmn \), 2\(^{nd}\) level \( cmn/2 \)
- Time for all i-th level computations \( cmn/2^{i-1} \) (each subproblem has \( n/2^i \) rows, the cols of all subprobs are distinct)
- Total time: \( \sum_{i=1}^{\log n} cmn/2^{i-1} \leq 2cmn \)
- Total space: \( O(m+n) \)
Dan Hirschberg

Daniel S. Hirschberg is a full professor in Computer Science at University of California, Irvine. His research interests are in the theory of design and analysis of algorithms.

End-Space Free Alignment

• Suppose spaces at the beginning and the end of the alignment contribute zero weight.

• **Example:**

  \[
  S = - - c a c - d b d v l \\
  T = l t c a b d d b - -
  \]

  No weight  No weight

• **Motivation:** “shotgun sequence assembly” - finding the original sequence given many of its subsequences (possibly overlapping).
End-Space Free Alignment (2)

- **solution**: similar to global alignment alg:

  - **Base conditions**: 
    \[ V(i,0) = 0 \]
    \[ V(0,j) = 0 \]
  - **Recurrence relation**: 
    \[ V(i,j) = \max \left\{ \begin{array}{c} V(i-1,j-1) + \sigma(s_i,t_j) \\
                               V(i-1,j) + \sigma(s_i,-) \\
                               V(i,j-1) + \sigma(-,t_j) \end{array} \right\} \]
  - **Search for** \(i^*\) and \(j^*\) such that 
    \[ V(n, i^*) = \max_i \{ V(n, i) \} \]
    \[ V(j^*, m) = \max_j \{ V(j, m) \} \]
  - **V(S, T) = \max \{ V(n, i^*), V(j^*, m) \}**

  - **Instead of**: \( \sum_{k=0..i} \sigma(s_k,-) \) and \( \sum_{k=0..j} \sigma(-,t_k) \)
  - The same as in global alignment

  - **Time complexity**: \( O(nm) \)
    - computing the matrix: \( O(nm) \),
    - finding \(i^*\) and \(j^*\): \( O(n+m) \).
  - **Space complexity**: for opt value: \( O(n+m) \)
    - computing the matrix: \( O(n+m) \),
    - computing \(i^*\) and \(j^*\) requires the last row and column to be saved: \( O(n+m) \)

  - Instead of \( V(n,m) \)
Why compare sequences? (II)

**Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor**

RF Doolittle, MW Hunkapiller, LE Hood, SG Devare, KC Robbins, SA Aaronson, HN Antoniades

**ABSTRACT**

The transforming protein of a primate sarcoma virus and a platelet-derived growth factor are derived from the same or closely related cellular genes. This conclusion is based on the demonstration of extensive sequence similarity between the transforming protein derived from the simian sarcoma virus onc gene, v-sis, and a human platelet-derived growth factor. The mechanism by which v-sis transforms cells could involve the constitutive expression of a protein with functions similar or identical to those of a factor active transiently during normal cell growth.
Local Alignment

**Definition:** Given sequences $S$, $T$, find subsequences $\alpha$ of $S$ and $\beta$ of $T$, of maximum similarity (i.e., with optimal global alignment between $\alpha$ & $\beta$).

**Motivation:**
- ignore stretches of non-coding DNA
- protein domains (functional subunits)

**Example:**
- $S=abcx\text{d}e\text{x}$, $T=\text{x}xx\text{c}d\text{d}e\text{d}$,
- Similarity score: 2 per match, -1 for subs/indel,
- $\alpha=\text{cx}d\text{e}$ and $\beta=\text{c-}d\text{e}$ have optimal alignment score.
Local alignments in the alignment graph
Computing Local Alignment

The **local suffix alignment** problem for $S'$, $T'$: find a (possibly empty) suffix $\alpha$ of $S'=s_1...s_i$ and a (possibly empty) suffix $\beta$ of $T'=t_1...t_j$ such that the value of their alignment is maximum over all values of alignments of suffixes of $S'$ and $T'$.

- $V(i,j)$: the value of optimal local suffix alignment for a given pair $i, j$ of indices.
- How are the $V(i,j)$ related to opt local alignment value?
Computing Local Alignment (2)

A scheme of the algorithm:

- **Assumption**: match ≥ 0, mismatch/indel ≤ 0
- Solve local suffix alignment for S' = s₁...sᵢ and T' = t₁...tⱼ by discarding prefixes whose similarity is ≤ 0
- Find the indices i*, j* after which the similarity only decreases.

Algorithm - Recursive Definition

**Base Condition**:
∀ i, j V(i, 0) = 0, V(0, j) = 0

**Recursion Step**: ∀ i > 0, j > 0

\[ V(i, j) = \max \begin{cases} 0, & \\
V(i-1, j-1) + \sigma(sᵢ, tⱼ), & \\
V(i, j-1) + \sigma(-, tⱼ), & \\
V(i-1, j) + \sigma(sᵢ, -) & \end{cases} \]

Compute i*, j*

s.t. \[ V(i*, j*) = \max_{1 \leq i \leq n, 1 \leq j \leq m} V(i, j) \]
\[
\begin{array}{cccccccccc}
\lambda & C & T & C & G & C & A & G & C \\
\hline
\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
A & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
T & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
T & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & 1 & 0 & 2 & 0 & 1 & 0 & 0 & 1 \\
A & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 \\
C & 0 & 1 & 0 & 1 & 0 & 2 & 0 & 1 & 1 \\
\end{array}
\]

+1 for a match, -1 for a mismatch, -5 for a space
Computing Local Alignment (3)

• Time $O(nm)$
• Space $O(n+m)$

The optimum value and the ends of subsequences $\alpha$ and $\beta$ can be found in linear space.
• Finding the starting point of the two subsequences can be done in linear space (ex.)
• The actual alignment can be computed using Hirschberg’s algorithm

• Smith-Waterman 81
Smith and Waterman
Gap Penalties

- **Observation**: spaces tend to occur in batches.

- **Idea**: when scoring an alignment, use the no. of contiguous gaps and not the no. of spaces.

- **Definitions**:
  - A *gap* is any maximal run of consecutive spaces in a single sequence of a given alignment.
  - The *length* of a gap is the number of spaces in it.
  - The number of gaps in the alignment is denoted by \#gaps.

- **Example**:
  - 4 gaps, 8 spaces, 7 matches, 0 mismatches.
Gap Penalty Models

Motivation:
- Indels of entire subsequence in a single mutation.
- When comparing cDNA to DNA, introns are gapped.

Constant Gap Penalty Model:
- Each individual space is free,
- Constant weight \( W_g \) for each gap, independent of its length (gap opening cost)

Goal: maximize \( \sum \sigma (s'_i, t'_i) + W_g \times \#gaps \)

Affine Gap Penalty Model:
- Additionally to \( W_g \), each space has cost \( W_s \). (gap extension cost)

Goal: max. \( \sum \sigma (s'_i, t'_i) + W_g \times \#gaps + W_s \times \#spaces \)
Alignment with Affine Gap Penalty

Three Types of Alignments:

1. \( S\ldots i \)
   \( T\ldots j \)
   - \( G(i,j) \) is max value of any alignment of type 1, where \( s_i \) and \( t_j \) match

2. \( S\ldots i \ldots \ldots \)\
   \( T\ldots \ldots \ldots \ldots j \)
   - \( E(i,j) \) is max value of any alignment of type 2, where \( t_j \) matches a space

3. \( S\ldots \ldots \ldots i \)
   \( T\ldots \ldots j \ldots \ldots \)
   - \( F(i,j) \) is max value of any alignment of type 3, where \( s_i \) matches a space
Alignment with Affine Gap Penalty (2)

**Base Conditions:**

\[ V(i, 0) = F(i, 0) = W_g + iW_s \]
\[ V(0, j) = E(0, j) = W_g + jW_s \]

**Recursive Computation:**

\[ V(i, j) = \max \{ E(i, j), F(i, j), G(i, j) \} \]

where:

- \( G(i, j) = V(i-1, j-1) + \sigma(s_i, t_j) \)
- \( E(i, j) = \max \{ E(i, j-1) + W_s, G(i, j-1) + W_g + W_s, F(i, j-1) + W_g + W_s \} \)
- \( F(i, j) = \max \{ F(i-1, j) + W_s, G(i-1, j) + W_g + W_s, E(i-1, j) + W_g + W_s \} \)

- Time complexity \( O(nm) \) - compute 4 matrices instead of one.
- Space complexity \( O(nm) \) - saving 4 matrices (trivial implementation).
Other Gap Penalty Models:

**Convex Gap Penalty Model:**
- Each additional space in a gap contributes less to the gap weight.
- **Example:** $W_g + \log(q)$, where $q$ is the length of the gap.
- solvable in $O(nm \log m)$ time

**Arbitrary Gap Penalty Model:**
- Most general gap weight.
- Weight of a gap is an arbitrary function of its length $w(q)$.
- solvable in $O(nm^2 + n^2m)$ time.