CS 466
Introduction to Bioinformatics
Lecture 5

Mohammed El-Kebir
September 9, 2020
Course Announcements

Instructor:
• Mohammed El-Kebir (melkebir)
• Office hours: Wednesdays, 3:15-4:15pm

TA:
• Aswhin Ramesh (aramesh7)
• Office hours: Fridays, 11:00-11:59am in SC 3405

**Homework 1:** Due on Sept. 18 (11:59pm)

**Midterm:** 10/4, 11-1pm @Transportation Building 103
(Conflict: 10/7, 7-9pm @Siebel 1302 -- to sign up email me)
Global Alignment problem: Given strings $v \in \Sigma^m$ and $w \in \Sigma^n$ and scoring function $\delta$, find alignment of $v$ and $w$ with maximum score. [Needleman-Wunsch algorithm]

Fitting Alignment problem: Given strings $v \in \Sigma^m$ and $w \in \Sigma^n$ and scoring function $\delta$, find an alignment of $v$ and a substring of $w$ with maximum global alignment score $s^*$ among all global alignments of $v$ and all substrings of $w$.

Local Alignment problem: Given strings $v \in \Sigma^m$ and $w \in \Sigma^n$ and scoring function $\delta$, find a substring of $v$ and a substring of $w$ whose alignment has maximum global alignment score $s^*$ among all global alignments of all substrings of $v$ and $w$. [Smith-Waterman algorithm]

Question: How to assess resulting algorithms?
Alignment is a path from source \((0, 0)\) to target \((m, n)\) in edit graph.

**Edit graph** is a weighed, directed grid graph \(G = (V, E)\) with source vertex \((0, 0)\) and target vertex \((m, n)\). Each edge \(((i, j), (k, l))\) has weight depending on direction.

Running time is \(O(mn)\)

[quadratic time]
**Time Complexity**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Edit graph* is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex $(m, n)$. Each edge $((i, j), (k, l))$ has weight depending on direction.

Alignment is a path from source $(0, 0)$ to target $(m, n)$ in edit graph.

Running time is $O(mn)$ [quadratic time]

**Question:** Compute alignment faster than $O(mn)$ time? [subquadratic time]
Space Complexity

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m = 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus, space complexity is $O(mn)$ [quadratic space]

Example:
To align a short read ($m = 100$) to human genome ($n = 3 \cdot 10^9$), we need 300 GB memory.

Size of DP table is $(m + 1) \times (n + 1)$
Space Complexity

Size of DP table is \((m + 1) \times (n + 1)\)

Thus, space complexity is \(O(mn)\) [quadratic space]

Example:
To align a short read \((m = 100)\) to human genome \((n = 3 \cdot 10^9)\), we need 300 GB memory.

Question: How long is an alignment?
Space Complexity

Size of DP table is \((m + 1) \times (n + 1)\)

Thus, space complexity is \(O(mn)\) [quadratic space]

Example:
To align a short read \(m = 100\) to human genome \(n = 3 \cdot 10^9\), we need 300 GB memory.

Question: How long is an alignment?

Question: Compute alignment in \(O(m)\) space? [linear space]
Outline

1. Recap of global, fitting, local and gapped alignment
2. Space-efficient alignment
3. Subquadratic time alignment

Reading:
• Jones and Pevzner. Chapters 7.1-7.4
• Lecture notes
Space Efficient Alignment

Computing $s[i, j]$ requires access to: $s[i-1, j]$, $s[i, j-1]$ and $s[i-1, j-1]$

$$s[i, j] = \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
 s[i-1, j] + \delta(v_i, -), & \text{if } i > 0, \\
 s[i, j-1] + \delta(-, w_j), & \text{if } j > 0, \\
 s[i-1, j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}$$

Figure 7.2 Calculating an alignment score requires no more than $2n$ space for an $n \times n$ alignment problem. Computing the alignment scores in each column requires only the scores in the preceding column. We show here the dynamic programming array—the data structure that holds the score at each vertex—instead of the graph.
Space Efficient Alignment

Computing $s[i, j]$ requires access to:
$s[i - 1, j]$, $s[i, j - 1]$ and $s[i - 1, j - 1]$

$$s[i, j] = \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
 s[i - 1, j] + \delta(v_i, -), & \text{if } i > 0, \\
 s[i, j - 1] + \delta(-, w_j), & \text{if } j > 0, \\
 s[i - 1, j - 1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}$$

Thus it suffices to store only two columns to compute optimal alignment score $s[m, n]$, i.e., $2(m + 1) = O(m)$ space.

Figure 7.2 Calculating an alignment score requires no more than $2n$ space for an $n \times n$ alignment problem. Computing the alignment scores in each column requires only the scores in the preceding column. We show here the dynamic programming array—the data structure that holds the score at each vertex—instead of the graph.
Space Efficient Alignment

Computing \( s[i, j] \) requires access to: 
\( s[i - 1, j], s[i, j - 1] \) and \( s[i - 1, j - 1] \)

\[
s[i, j] = \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
\ s[i - 1, j] + \delta(v_i, -), & \text{if } i > 0, \\
\ s[i, j - 1] + \delta(-, w_j), & \text{if } j > 0, \\
\ s[i - 1, j - 1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}
\]

Thus it suffices to store only two columns to compute optimal alignment score \( s[m, n] \), i.e., \( 2(m + 1) = O(m) \) space.

**Question:** What if we want alignment itself?
Space Efficient Alignment – First Attempt

- What if also want optimal alignment?
- **Easy**: keep best pointers as fill in table.
- **No!** Do not know which path to keep until computing recurrence at each step.

<table>
<thead>
<tr>
<th></th>
<th>(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v)</td>
<td></td>
</tr>
</tbody>
</table>
Space Efficient Alignment – First Attempt

• What if also want optimal alignment?
• **Easy**: keep best pointers as fill in table.
• **No!** Do not know which path to keep until computing recurrence at each step.
Space Efficient Alignment – First Attempt

• What if also want optimal alignment?
• **Easy**: keep best pointers as fill in table.
• **No!** Do not know which path to keep until computing recurrence at each step.

Best score for column might not be part of best alignment!
Space Efficient Alignment – Second Attempt

Alignment is a path from source \((0, 0)\) to target \((m, n)\) in edit graph

Maximum weight path from \((0,0)\) to \((m, n)\) passes through \((i^*, n/2)\)

**Question:** What is \(i^*\)?
Space Efficient Alignment – Second Attempt

\[ \frac{n}{2} \]

\[ i^* \]

\[ i \]

\[ m \]
Space Efficient Alignment – Second Attempt

\[ n/2 \]

\[ i^* \]

\[ i \]

\[ m \]
Hirschberg\((i, j, i', j')\)

1. \(\text{if } j' - j > 1 \)

2. \(i^* \leftarrow \arg \max_{i \leq i'' \leq i'} \text{wt}(i'')\)

3. Report \((i^*, j + \frac{j' - j}{2})\)

4. Hirschberg\((i, j, i^*, j + \frac{j' - j}{2})\)

5. Hirschberg\((i^*, j + \frac{j' - j}{2}, i', j')\)

**Figure 7.3** Space-efficient sequence alignment. The computational time (i.e., the area of the solid rectangles) decreases by a factor of 2 at every iteration.
Linear-Space Sequence Alignment

Hirschberg\((i, j, i', j')\)

1. if \(j' - j > 1\)
2. \(i^* \leftarrow \text{arg max}_{i \leq i'' \leq i'} \text{wt}(i'')\)
3. Report \((i^*, j + \frac{j' - j}{2})\)
4. Hirschberg\((i, j, i^*, j + \frac{j' - j}{2})\)
5. Hirschberg\((i^*, j + \frac{j' - j}{2}, i', j')\)

Time:
\[\text{area} + \text{area}/2 + \text{area}/4 + \ldots\]
\[= \text{area} \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right)\]
\[\leq 2 \times \text{area} = O(mn)\]

Space: \(O(m)\)

Figure 7.3  Space-efficient sequence alignment. The computational time (i.e., the area of the solid rectangles) decreases by a factor of 2 at every iteration.
Hirschberg\((i, j, i', j')\)

1. \(\text{if } j' - j > 1\)
2. \(i^* \leftarrow \arg \max_{i \leq i'' \leq i'} \text{wt}(i'')\)
3. \(\text{Report } (i^*, j + \frac{j' - j}{2})\)
4. \(\text{Hirschberg}(i, j, i^*, j + \frac{j' - j}{2})\)
5. \(\text{Hirschberg}(i^*, j + \frac{j' - j}{2}, i', j')\)

Time:
\[
\text{area} + \frac{\text{area}}{2} + \frac{\text{area}}{4} + \ldots \\
= \text{area} \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right) \\
\leq 2 \times \text{area} = \mathcal{O}(mn)
\]

Space: \(\mathcal{O}(m)\)

Question: How to reconstruct alignment from reported vertices?
Hirschberg Algorithm: Reversing Edges Necessary?

Max weight path from \((0,0)\) to \((m,n)\) through \((i^*,n/2)\)

\[
i^* = \arg \max_{0 \leq i \leq m} \{ \text{prefix}(i) + \text{suffix}(i) \}
\]

Compute \{prefix\(i) \mid 0 \leq i \leq m\} in \(O mj\) time and \(O(m)\) space, by starting from \((0,0)\) to \((m, j)\) keeping only two columns in memory. [single-source multiple destinations]
Hirschberg Algorithm: Reversing Edges Necessary?

Max weight path from (0,0) to (m,n) through (i*, n/2)

\[ i^* = \arg \max_{0 \leq i \leq m} \{ \text{prefix}(i) + \text{suffix}(i) \} \]

Compute \{prefix(i) | 0 \leq i \leq m\} in O(mj) time and O(m) space, by starting from (0,0) to (m,j) keeping only two columns in memory. [single-source multiple destinations]

Want: Compute \{suffix(i) | 0 \leq i \leq m\} in O(mj) time and O(m) space

Doing a longest path from each \((i,j)\) to \((m,n)\) (for all \(0 \leq i \leq m\)) will not achieve desired running time!

Reversing edges enables single-source multiple destination computation in desired time and space bound!
Problem: Given reported vertices and scores \{i_0, 0, s_0\}, ..., \{i_n, n, s_n\}, find intermediary vertices.

Hirschberg(i, j, i', j')

1. if \(j' - j > 1\)
2. \(i^* \leftarrow \arg \max_{0 \leq i \leq m} \text{wt}(i)\)
3. Report \((i^*, j + \frac{j' - j}{2})\)
4. Hirschberg\((i, j, i^*, j + \frac{j' - j}{2})\)
5. Hirschberg\((i^*, j + \frac{j' - j}{2}, i', j')\)

Transposing matrix does not help, because gaps could occur in both input sequences (and there might be multiple opt. alignments)
Linear Space Alignment – The Hirschberg Algorithm
Outline

1. Recap of global, fitting, local and gapped alignment
2. Space-efficient alignment
3. Subquadratic time alignment

Reading:
• Jones and Pevzner. Chapters 7.1-7.4
• Lecture notes
Banded Alignment

Constraint path to band of width $k$ around diagonal

Running time: $O(nk)$

Gives a good approximation of highly identical sequences

Constrain traceback to band of DP matrix (penalize big gaps)

Figure source: [http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf](http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf)
Banded Alignment

Constraint path to band of width $k$ around diagonal

Running time: $O(nk)$

Gives a good approximation of highly identical sequences

**Question:** How to change recurrence to accomplish this?

Constrain traceback to band of DP matrix (penalize big gaps)

Figure source: [http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf](http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf)
Block Alignment

Divide input sequences into blocks of length $t$

\[
\begin{align*}
\begin{array}{c}
\quad v_1, \ldots, v_t \\
\quad v_{t+1}, \ldots, v_{2t} \\
\quad \quad \ldots \\
\quad v_{m-t+1}, \ldots, v_m \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\quad w_1, \ldots, w_t \\
\quad w_{t+1}, \ldots, w_{2t} \\
\quad \quad \ldots \\
\quad w_{n-t+1}, \ldots, w_n \\
\end{array}
\end{align*}
\]
Block Alignment

Divide input sequences into blocks of length $t$

$$v_1, \ldots, v_t \quad v_{t+1}, \ldots, v_{2t} \quad \ldots \quad v_{m-t+1}, \ldots, v_m$$

$$w_1, \ldots, w_t \quad w_{t+1}, \ldots, w_{2t} \quad \ldots \quad w_{n-t+1}, \ldots, w_n$$

Require that paths in edit graph pass through **corners** of blocks.

*Figure 7.4* Two paths in a $40 \times 40$ grid partitioned into 16 subgrids of size $10 \times 10$. The black path (a) is a block path, while the gray path (b) is not.
### Block Alignment

**Divide input sequences into blocks of length $t$**

\[
\begin{align*}
  v_1, \ldots, v_t & \quad v_{t+1}, \ldots, v_{2t} & \quad \ldots & \quad v_{m-t+1}, \ldots, v_m \\
  w_1, \ldots, w_t & \quad w_{t+1}, \ldots, w_{2t} & \quad \ldots & \quad w_{n-t+1}, \ldots, w_n
\end{align*}
\]

**Require that paths in edit graph pass through corners of blocks**

\[
\begin{align*}
s[i, j] &= \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
  s[i - 1, j] - \sigma, & \text{if } i > 0, \\
  s[i, j - 1] - \sigma, & \text{if } j > 0, \\
  s[i - 1, j - 1] + \beta(i, j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}
\end{align*}
\]

**Figure 7.4** Two paths in a $40 \times 40$ grid partitioned into 16 subgrids of size $10 \times 10$. The black path (a) is a block path, while the gray path (b) is not.

$0 \leq i, j \leq t$ and $\beta(i, j)$ is max score alignment between block $i$ of $v$ and block $j$ of $w$. 

Copyright © 2004. MIT Press. All rights reserved. May not be reproduced in any form without permission from the publisher, except fair uses permitted under U.S. or applicable copyright law.
Block Alignment – First Attempt: Pre-compute $\beta(i, j)$

$0 \leq i, j \leq n/t$ and $\beta(i, j)$ is max score alignment between block $i$ of $v$ and block $j$ of $w$

$s[i, j] = \max\begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
 s[i-1, j] - \sigma, & \text{if } i > 0, \\
 s[i, j-1] - \sigma, & \text{if } j > 0, \\
 s[i-1, j-1] + \beta(i, j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}$
Block Alignment – First Attempt: Pre-compute $\beta(i, j)$

$0 \leq i, j \leq n/t$ and $\beta(i, j)$ is max score alignment between block $i$ of $v$ and block $j$ of $w$

$$s[i, j] = \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
\sigma, & \text{if } i > 0, \\
\sigma, & \text{if } j > 0, \\
\underline{s[i - 1, j - 1]} + \beta(i, j), & \text{if } i > 0 \text{ and } j > 0.
\end{cases}$$

**Question:**
How much time to compute all $\beta(i, j)$?
Block Alignment – First Attempt: Pre-compute $\beta(i, j)$

$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1, j] - \sigma, & \text{if } i > 0, \\ s[i, j-1] - \sigma, & \text{if } j > 0, \\ s[i-1, j-1] + \beta(i, j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$

**Question:**
How much time to compute all $\beta(i, j)$?

Computing $\beta(i, j)$ takes $O(t^2)$ time.

There are $n/t \times n/t$ values $\beta(i, j)$.

Total: $O\left(\frac{n}{t} \times \frac{n}{t} \times t^2\right) = O(n^2)$ time.
Block Alignment – Four Russians Technique

Pre-compute and store all $\beta_{ij}$

Pre-compute and store all max weight alignments $S[\mathbf{v}', \mathbf{w}']$ of all pairs $(\mathbf{v}', \mathbf{w}')$ of length $t$ strings

Algorithm:
1. Precompute $S[\mathbf{v}', \mathbf{w}']$ where $\mathbf{v}', \mathbf{w}' \in \Sigma^t$
2. Compute block alignment between $\mathbf{v}$ and $\mathbf{w}$ using $S$

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] - \sigma, & \text{if } i > 0, \\ s[i, j - 1] - \sigma, & \text{if } j > 0, \\ s[i - 1, j - 1] + S[v(i), w(j)], & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$
Block Alignment – Four Russians Technique

![Diagram of block alignment](image)

**Algorithm:**

1. Precompute $S[v', w']$ where $v', w' \in \Sigma^t$
2. Compute block alignment between $v$ and $w$ using $S$

**Pre-compute and store all $\beta_{ij}$**

Pre-compute and store *all* max weight alignments $S[\mathbf{v}', \mathbf{w}']$ of *all* pairs $(\mathbf{v}', \mathbf{w}')$ of length $t$ strings

**Question:** How to choose $t$ for DNA?

\[
s[i, j] = \max \begin{cases} 
0, & \text{if } i = 0 \text{ and } j = 0, \\
 s[i - 1, j] - \sigma, & \text{if } i > 0, \\
 s[i, j - 1] - \sigma, & \text{if } j > 0, \\
 s[i - 1, j - 1] + S[v(i), w(j)], & \text{if } i > 0 \text{ and } j > 0.
\end{cases}
\]
Block Alignment – Four Russians Technique

**Question:** How to choose $t$ for DNA?
Fastest Subquadratic Alignment* Algorithm

Edit distance in \( O(n^2 / \log n) \) time

Barely subquadratic!

Want: \( O(n^{2-\varepsilon}) \) time where \( \varepsilon > 0 \)

*for edit distance
Fastest Subquadratic Alignment* Algorithm

Edit distance in $O(n^2 / \log n)$ time

Barely subquadratic!

Want: $O(n^{2-\varepsilon})$ time where $\varepsilon > 0$

Question: Is $n^{2-\varepsilon}$ in $O(n^2 / \log n)$ for any $\varepsilon > 0$?
Hardness Result for Edit Distance [STOC 2015]

Edit Distance Cannot Be Computed in strongly subquadratic time
(unless SETH is false)

Arturs Backurs
MIT

Piotr Indyk
MIT

Abstract

The edit distance (a.k.a. the Levenshtein distance) between two strings is defined as the minimum number of insertions, deletions or substitutions of symbols needed to transform one string into another. The problem of computing the edit distance between two strings is a classical computational task, with a well-known algorithm based on dynamic programming. Unfortunately, all known algorithms for this problem run in nearly quadratic time.

In this paper we provide evidence that the near-quadratic running time bounds known for the problem of computing edit distance might be tight. Specifically, we show that, if the edit distance can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then the satisfiability of conjunctive normal form formulas with $N$ variables and $M$ clauses can be solved in time $M^{O(1)}2^{(1-\epsilon)N}$ for a constant $\epsilon > 0$. The latter result would violate the Strong Exponential Time Hypothesis, which postulates that such algorithms do not exist.

$O(n^{2-\epsilon})$ time where $\epsilon > 0$
For 40 years, computer scientists looked for a solution that doesn’t exist [1]

For 40 years, computer scientists have tried in vain to find a faster way to do an important calculation known as “edit distance.” Thanks to groundbreaking work from two researchers at MIT, they now know the reason they’ve continually failed is because a faster method is actually impossible to create.

In biology n does not go to infinity [2]

I recently read a “brainiac” column in the Boston Globe titled “For 40 years, computer scientists looked for a solution that doesn’t exist” that caused me to facepalm so violently I now have pain in my right knee.

Take Home Messages

1. Global alignment in $O(mn)$ time and $O(m)$ space
   • Hirschberg algorithm

2. Block alignment can be done in subquadratic time
   • Four Russians Technique: $O(n^2 / \log n)$ time

3. Global alignment cannot be done in $O(n^{2-\varepsilon})$ time under SETH

Reading:
• Jones and Pevzner. Chapters 7.1-7.4
• Lecture notes