CSE 427
Computational Biology
Winter 2008

Sequence Alignment;
DNA Replication

Sequence Alignment

Part I
Motivation, dynamic programming,
global alignment

Sequence Alignment

• What
• Why
• A Simple Algorithm
• Complexity Analysis
• A better Algorithm:
  “Dynamic Programming”
Sequence Similarity: What

G G A C C
T A C T A A G
| :   |  :  |   |  :
T C C – A A T

Sequence Similarity: Why

• Most widely used comp. tools in biology
• New sequence always compared to sequence data bases
  Similar sequences often have similar origin or function
• Recognizable similarity after $10^8$ – $10^9$ yr

BLAST Demo

Terminology
(CS, not necessarily Bio)

• String: ordered list of letters
  TATAAG
• Prefix: consecutive letters from front
  empty, T, TA, TAT, ...
• Suffix: ... from end
  empty, G, AG, AAG, ...
• Substring: ... from ends or middle
  empty, TAT, AA, ...
• Subsequence: ordered, nonconsecutive
  TT, AAA, TAG, ...
**Sequence Alignment**

\[
\begin{array}{c}
\text{a c b c d b} \\
\text{c a d b} \\
\end{array}
\quad
\begin{array}{c}
\text{a c} \\
\text{b c d b} \quad \text{–} \\
\text{c a d b} \quad \text{–} \\
\text{–} \\
\end{array}
\]

**Defn:** An alignment of strings S, T is a pair of strings S', T' (with spaces) s.t.

1. \(|S'| = |T'|\), and \((|S| = \text{"length of S"})\)
2. removing all spaces leaves S, T

**Optimal Alignment: A Simple Algorithm**

```
for all subseqs A of S, B of T s.t. |A| = |B| do
  align A[i] with B[i], 1 \leq i \leq |A|
  align all other chars to spaces
  compute its value
  retain the max
end
```

Output the retained alignment

---

**Alignment Scoring**

```
\begin{array}{c}
\text{a c b c d b} \\
\text{c a d b} \quad \text{–} \\
\text{–} \\
\end{array}
\quad
\begin{array}{c}
\text{a c} \\
\text{b c d b} \quad \text{–} \\
\text{c a d b} \quad \text{–} \\
\text{–} \\
\end{array}
```

- The score of aligning (characters or spaces) x & y is \(\sigma(x, y)\).
- Value of an alignment \(\sum_{i=1}^{S'} \sigma(S'[i], T'[i])\)
- An optimal alignment: one of max value

**Analysis**

- Assume \(|S| = |T| = n\)
- Cost of evaluating one alignment: \(\geq n\)
- How many alignments are there: \(\geq \binom{2n}{n}\)
  - pick \(n\) chars of S, T together
  - say \(k\) of them are in S
  - match these \(k\) to the \(k\) unpicked chars of T
- Total time: \(\geq n \binom{2n}{n} > 2^{2n}\), for \(n > 3\)
- E.g., for \(n = 20\), time is > 2^{40} operations
Polynomial vs Exponential Growth

Asymptotic Analysis

- How does run time grow as a function of problem size?
  - \( n^2 \) or \( 100 n^2 + 100 n + 100 \) vs \( 2^{2n} \)
- **Defn:** \( f(n) = O(g(n)) \) iff there is a constant \( c \) s.t. \( |f(n)| \leq cg(n) \) for all sufficiently large \( n \).
  - \( 100 n^2 + 100 n + 100 = O(n^2) \) [e.g. \( c = 300 \), or \( 101 \)]
  - \( n^2 = O(2^{2n}) \)
  - \( 2^{2n} \) is not \( O(n^2) \)

Utility of Asymptotics

- “All things being equal,” smaller asymptotic growth rate is better
- All things are never equal
- Even so, big-O bounds often let you quickly pick most promising candidates among competing algorithms
- Poly time algorithms often practical; non-poly algorithms seldom are.
  (Yes, there are exceptions.)

Fibonacci Numbers

```java
fib(n) {
  if (n <= 1) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2); // Simple recursion, but many repeated subproblems!
  }
}
```

Time = \( \Omega(1.61^n) \)
Fibonacci, II

```c
int fib[n];
fib[0] = 1;
fib[1] = 1;
for(i=2; i<=n; i++) {
fib[i] = fib[i-1] + fib[i-2];
}
return fib[n];
```

“Dynamic Programming”
Avoid repeated work by tabulating solutions to repeated subproblems

\[ \text{Time} = O(n) \]
(in this case)

Candidate for Dynamic Programming?

- Common Subproblems?
  - Plausible: probably re-considering alignments of various small substrings unless we're careful.
- Optimal Substructure?
  - Plausible: left and right "halves" of an optimal alignment probably should be optimally aligned (though they obviously interact a bit at the interface).
  - (Both made rigorous below.)

Optimal Substructure
(In More Detail)

- Optimal alignment *ends* in 1 of 3 ways:
  - last chars of S & T aligned with each other
  - last char of S aligned with space in T
  - last char of T aligned with space in S
    - (never align space with space; \( \sigma(-,-) < 0 \))
- In each case, the *rest* of S & T should be optimally aligned to each other

Optimal Alignment in \( O(n^2) \) via “Dynamic Programming”

- Input: S, T, |S| = n, |T| = m
- Output: value of optimal alignment

Easier to solve a “harder” problem:

\[
V(i,j) = \begin{cases} 
\text{value of optimal alignment of } S[1], \ldots, S[i] \text{ with } T[1], \ldots, T[j] 
\text{ for all } 0 \leq i \leq n, 0 \leq j \leq m. 
\end{cases}
\]
### Base Cases

- **V(i,0):** first i chars of S all match spaces
  \[
  V(i,0) = \sum_{k=1}^{i} \sigma(S[k],-)
  \]
- **V(0,j):** first j chars of T all match spaces
  \[
  V(0,j) = \sum_{k=1}^{j} \sigma(-,T[k])
  \]

### General Case

Opt align of \(S[1], \ldots, S[i]\) vs \(T[1], \ldots, T[j]\):

\[
V(i,j) = \max \left\{ V(i-1,j-1) + \sigma(S[i],T[j]), V(i-1,j) + \sigma(S[i],-), V(i,j-1) + \sigma(-,T[j]) \right\}
\]

for all \(1 \leq i \leq n, 1 \leq j \leq m\).

### Calculating One Entry

\[
V(i,j) = \max \left\{ V(i-1,j-1) + \sigma(S[i],T[j]), V(i-1,j) + \sigma(S[i],-), V(i,j-1) + \sigma(-,T[j]) \right\}
\]

### Example

Mismatch = -1
Match = 2

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<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>1</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>

Time = O(mn)
Example

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

Mismatch = -1
Match = 2

Finding Alignments: Trace Back

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

Complexity Notes

- Time = $O(mn)$, (value and alignment)
- Space = $O(mn)$
- Easy to get value in Time = $O(mn)$ and Space = $O(\min(m,n))$
- Possible to get value and alignment in Time = $O(mn)$ and Space = $O(\min(m,n))$ but tricky.

Sequence Alignment

Part II
Local alignments & gaps
Variations

- Local Alignment
  - Preceding gives global alignment, i.e. full length of both strings;
  - Might well miss strong similarity of part of strings amidst dissimilar flanks
- Gap Penalties
  - 10 adjacent spaces cost $10 \times$ one space?
  - Many others

Local Alignment: Motivations

- “Interesting” (evolutionarily conserved, functionally related) segments may be a small part of the whole
  - “Active site” of a protein
  - Scattered genes or exons amidst “junk”, e.g. retroviral insertions, large deletions
  - Don’t have whole sequence
- Global alignment might miss them if flanking junk outweighs similar regions

Local Alignment

Optimal local alignment of strings S & T:
Find substrings A of S and B of T having max value global alignment

$$S = \text{abcxdex} \quad A = \text{c x d e}$$
$$T = \text{xxxcde} \quad B = \text{c - d e} \quad \text{value = 5}$$

The “Obvious” Local Alignment Algorithm

```plaintext
for all substrings A of S and B of T
  Align A & B via dynamic programming
  Retain pair with max value
end;

Output the retained pair
```

Time: $O(n^2)$ choices for A, $O(m^2)$ for B,
$O(nm)$ for DP, so $O(n^2m^2)$ total.
[Best possible? Lots of redundant work…]
Local Alignment in O(nm) via Dynamic Programming

- Input: S, T, |S| = n, |T| = m
- Output: value of optimal local alignment

Better to solve a “harder” problem for all 0 ≤ i ≤ n, 0 ≤ j ≤ m:

\[ V(i,j) = \max \text{ value of opt (global) alignment of a suffix of S[1], ..., S[i] with a suffix of T[1], ..., T[j]} \]

Report best i,j

Base Cases

- Assume \( \sigma(x, -) \leq 0, \sigma(-, x) \leq 0 \)
- \( V(i, 0) \): some suffix of first i chars of S; all match spaces in T; best suffix is empty
  \[ V(i, 0) = 0 \]
- \( V(0, j) \): similar
  \[ V(0, j) = 0 \]

General Case Recurrences

Opt suffix align S[1], ..., S[i] vs T[1], ..., T[j]:

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

for all 1 ≤ i ≤ n, 1 ≤ j ≤ m.

Scoring Local Alignments

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finding Local Alignments

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes

- Time and Space = O(mn)
- Space O(min(m,n)) possible with time O(mn), but finding alignment is trickier
- Local alignment: “Smith-Waterman”
- Global alignment: “Needleman-Wunsch”

Alignment With Gap Penalties

- **Gap**: maximal run of spaces in S’ or T’
  
  \[
  \text{ab----c-d} \\
  \text{a-ddddcbd} \\
  \text{2 gaps in S’, 1 in T’}
  \]
- Motivations, e.g.:
  - mutation might insert/delete several or even many residues at once
  - matching cDNA (no introns) to genomic DNA (exons and introns)
  - Some parts of proteins less critical

A Protein Structure:
(Dihydrofolate Reductase)
Sequence Evolution

Nothing in Biology Makes Sense Except in the Light of Evolution  
Theodosius Dobzhansky, 1973

Changes happen at random  
Deleterious/neutral/advantageous changes  
unlikely/possibly/likely spread widely in a population  
Changes are less likely to be tolerated in positions  
involved in many/close interactions, e.g.  
enzyme binding pocket  
protein/protein interaction surface  
...

Gap Penalties

- Score = f(gap length)
- Kinds, & best known alignment time
  - general $O(n^3)$
  - convex $O(n^2 \log n)$
  - affine $O(mn)$

Global Alignment with Affine Gap Penalties

$V(i,j) =$ value of opt alignment of $S[1], ..., S[i]$ with $T[1], ..., T[j]$  
$G(i,j) =$ ..., s.t. last pair matches $S[i] & T[j]$  
$F(i,j) =$ ..., s.t. last pair matches $S[i] &$  
$E(i,j) =$ ..., s.t. last pair matches $- & T[j]$  

Time: $O(mn)$  [calculate all, $O(1)$ each]
**Affine Gap Algorithm**

Gap penalty = \( g + s \cdot (\text{gap length}) \), \( g,s \geq 0 \)

\[ V(i,0) = E(i,0) = V(0,i) = F(0,i) = -g - i \cdot s \]

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

\[ G(i,j) = V(i-1,j-1) + \sigma(S[i],T[j]) \]

\[ F(i,j) = \max(F(i-1,j)-s, V(i-1,j)-g-s) \]

\[ E(i,j) = \max(E(i,j-1)-s, V(i,j-1)-g-s) \]

**Summary**

- Functionally similar proteins/DNA often have recognizably similar sequences even after eons of divergent evolution
- Ability to find/compare/experiment with "same" sequence in other organisms is a huge win
- Surprisingly simple scoring model works well in practice: score each position separately & add, possibly w/ fancier gap model like affine
- Simple “dynamic programming” algorithms can find optimal alignments under these assumptions in poly time (product of sequence lengths)
- This, and heuristic approximations to it like BLAST, are workhorse tools in molecular biology

**Weekly Bio Interlude**

**DNA Replication**
Issues & Complications, I

- 1st ~10 nt’s added are called the primer
- In simple model, DNA pol has 2 jobs: prime & extend
- Priming is error-prone
- So, specialized primase does the priming; pol specialized for fast, accurate extension
- Still doesn’t solve the accuracy problem (hint: primase makes an RNA primer)

Issue 2: Rep Forks & Helices

- "Replication Fork": DNA double helix is progressively unwound by a DNA helicase, and both resulting single strands are duplicated
- DNA polymerase synthesizes new strand 5' -> 3' (reading its template strand 3' -> 5')
- That means on one (the "leading") strand, DNA pol is chasing/pushing the replication fork
- But on the other "lagging" strand, DNA pol is running away from it.

Issue 3: Fragments

- Lagging strand gets a series of “Okazaki fragments” of DNA (~200nt in eukaryotes) following each primer
- The RNA primers are later removed by a nuclease and DNA pol fills gaps (more accurate than primase)
- Fragments joined by ligase

Issue 4: Coord Lead/Lag

Alberts et al., Mol. Biol. of the Cell, 3rd ed., p258
Issue 5: Twirls & Tangles

- Unwinding helix (~10 nucleotides per turn) would cause stress. *Topoisomerase I* cuts DNA backbone on one strand, allowing it to spin about the remaining bond, relieving stress.
- *Topoisomerase II* can cut & rejoin both strands, after allowing another double strand to pass through the gap, de-tangling it.

Issue 6: Proofreading

- Error rate of pol itself is $\sim 10^{-4}$, but overall rate is $10^{-9}$, due to proofreading & repair, e.g.
  - pol itself can back up & cut off a mismatched base if one happens to be inserted
  - priming the new strand is hard to do accurately, hence RNA primers, later removed & replaced
  - other enzymes scan helix for “bulges” caused by base mismatch, figure out which strand is original, cut away new (faulty) copy; DNA pol fills gap
- which strand is original? In bacteria, some A’s are “methylated”, but not immediately after replication

Replication Summary

- Speed: 50 (eukaryotes) - 500 (prokaryotes) bp/sec
- Accuracy: 1 error per $10^9$ bp
- Complex & highly optimized
- Highly similar across all living cells

More info: Alberts et al., Mol. Biol. of the Cell