**Sequence Alignment and Dynamic Programming**

**Reminder: Last lecture / recitation**
- Schedule for the term
  - 'Foundations' till midterm
  - 'Frontiers' lead to final project
  - Duality: basic problems / fundamental techniques
- Biology introduction
  - DNA, RNA, protein, transcription, translation
  - Why computational biology
- First problem: Motif discovery
  - Counting motif instances across the genome
  - Counting conserved motif instances
  - Problem set: discover motifs in actual yeast genome

**Today's goal:**
How do we actually align two genes?
Genomes change over time

begin
ACGTCACT
ACGTAC
ACGTCA
ACTCA

mutation
deletion

insertion
end
TAGTGCA

Goal of alignment: Infer edit operations

begin
ACGTCACT

mutation

deletion

insertion
end
TAGTGCA

Question 1: Aligning two (ungapped) strings

• Given two possibly related strings S1 and S2
  – What is the longest common substring? (no gaps)

S1: ACGTCACT
S2: TAGTGCA

Scoring function:
Match(x,x) = +1
Mismatch(A,G) = -½
Mismatch(C,T) = -½
Mismatch(x,y) = -1

S1

ACGTCACT

offset: +1

S2

TAGTGCA

S1

ACGTCACT

offset: -2

S2

TAGTGCA

Q2: Aligning two (possibly gapped) sequences

• Given two possibly related strings S1 and S2
  – What is the longest common subsequence? (gaps allowed)

S1: ACGTCACT
S2: TAGTGCA

Can we simply enumerate all possible alignments?

• Ways to align two sequences of length m, n
\[
\binom{n+m}{m} = \frac{(m+n)!}{(m!)^2} \approx \frac{2^{m+n}}{\sqrt{\pi \cdot m}}
\]

• For two sequences of length n

<table>
<thead>
<tr>
<th>n</th>
<th>Enumeration</th>
<th>Today's lecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>184,756</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>1.40E+11</td>
<td>400</td>
</tr>
<tr>
<td>100</td>
<td>9.00E+58</td>
<td>10,000</td>
</tr>
</tbody>
</table>
Key insight: score is additive!

• Compute best alignment recursively
  – For a given aligned pair \((i, j)\), the best alignment is:
    • Best alignment of \(S_1[1..i]\) and \(S_2[1..j]\)
    • + Best alignment of \(S_1[i..n]\) and \(S_2[j..m]\)

Solution #1 – Memoization

• Create a big dictionary, indexed by aligned seqs
  – When you encounter a new pair of sequences
  – If it is in the dictionary:
    • Look up the solution
  – If it is not in the dictionary
    • Compute the solution
    • Insert the solution in the dictionary
  • Ensures that there is no duplicated work
    – Only need to compute each sub-alignment once!

Solution #2 – Dynamic programming

• Create a big table, indexed by \((i,j)\)
  – Fill it in from the beginning all the way till the end
  – You know that you’ll need every subpart
  – Guaranteed to explore entire search space
  • Ensures that there is no duplicated work
    – Only need to compute each sub-alignment once!
  • Very simple computationally!

A simple introduction to Dynamic Programming

• Fibonacci numbers

Fibonacci numbers are ubiquitous in nature
Computing Fibonacci numbers: Top down

- **Goal:** Compute the nth Fibonacci number.
  - \( F(0) = 1, F(1) = 1, F(n) = F(n-1) + F(n-2) \)
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...
- **Top-down approach:**
  - **Python code**
    ```python
    def fibonacci(n):
        if n==1 or n==2: return 1
        return fibonacci(n-1) + fibonacci(n-2)
    ```
  - Analysis: \( T(n) = T(n-1) + T(n-2) = (\ldots) = O(2^n) \)

Computing Fibonacci numbers: Bottom up

- **Top-down approach**
  - **Python code**
  - Analysis: \( T(n) = O(n) \)
    ```python
    def fibonacci(n):
        fib_table[1] = 1
        fib_table[2] = 1
        for i in range(3,n+1):
            fib_table[i] = fib_table[i-1]+fib_table[i-2]
        return fib_table[n]
    ```
- Analysis: \( T(n) = O(n) \)

What have we learned?

- **Principles of dynamic programming**
  - Reveal identical sub-problems
  - Order computation to enable result reuse
  - Systematically fill in table of results
  - Express larger problems from their subparts
- **Ordering of computations matters**
  - Naïve top-down approach very slow
    - results of smaller problems not available
    - repeated work
  - Systematic bottom-up approach successful
    - Systematically solve each sub-problem
    - Fill-in table of sub-problem results in order
    - Look up solutions instead of recomputing.

How do we apply dynamic programming to sequence alignment?

**Key insight:** score is additive!

- **Compute best alignment recursively**
  - For a given aligned pair \((i, j)\), the best alignment is:
    - Best alignment of \(S[1..i]\) and \(S[1..j]\)
    - Best alignment of \(S[1..i]\) and \(S[1..j]\)

Store score of aligning \((i, j)\) in matrix \(M(i, j)\)

- Best alignment ⇒ Best path through the matrix
Filling in the dynamic programming matrix

- **Local update rules**:  
  - Compute next alignment based on previous alignment  
  - Just like Fibonacci numbers: $F[i] = F[i-1] + F[i-2]$  
  - Table lookup!

---

**Duality: seq. alignment ↔ path through the matrix**

**Goal:** Find best path through the matrix

---

**0. Setting up the scoring matrix**

- **Initialization**:  
  - Top right: 0  

- **Update Rule**:  
  - $A(i,j) = \max\{\ldots\}$

- **Termination**:  
  - Bottom right

---

**1. Allowing gaps in s**

- **Initialization**:  
  - Top right: 0

- **Update Rule**:  
  - $A(i,j) = \max\{\ldots\}$

- **Termination**:  
  - Bottom right

---

**2. Allowing gaps in t**

- **Initialization**:  
  - Top right: 0

- **Update Rule**:  
  - $A(i,j) = \max\{\ldots\}$

- **Termination**:  
  - Bottom right

---

**3. Allowing mismatches**

- **Initialization**:  
  - Top right: 0

- **Update Rule**:  
  - $A(i,j) = \max\{\ldots\}$

- **Termination**:  
  - Bottom right
4. Choosing optimal paths

Initialization:
- Top right: 0

Update Rule:
\[ A(i,j) = \max \{ \]
- \[ A(i-1, j) - 2 \]
- \[ A(i, j-1) - 2 \]
- \[ A(i-1, j-1) - 1 \]
\]

Termination:
- Bottom right

5. Rewarding matches

Initialization:
- Top right: 0

Update Rule:
\[ A(i,j) = \max \{ \]
- \[ A(i-1, j) - 2 \]
- \[ A(i, j-1) - 2 \]
- \[ A(i-1, j-1) \pm 1 \]
\]

Termination:
- Bottom right

What is missing?
- We know how to compute the best score
  - Simply the number at the bottom right entry
- But we need to remember where it came from
  - Pointer to the choice we made at each step
- Retrace path through the matrix
  - Need to remember all the pointers

Time needed: \( O(m \times n) \)
Space needed: \( O(m \times n) \)

Can we do better than that?

Bounded Dynamic Programming

Initialization:
- \( F(i,0), F(0,j) \) undefined for \( i, j > k \)

Iteration:
For \( i = 1 \ldots M \)
For \( j = \max(1, i - k(\mathbf{N})) \ldots \min(N, i+k(\mathbf{N})) \)

\[ F(i-1, j-1) + s(x_i, y_j) \]
\[ F(i, j) = \max \{ F(i, j-1) - d, F(i-1, j) - d \} \]

Termination: same

Linear space alignment

It is easy to compute \( F(M, N) \) in linear space

Allocate (column[1])
Allocate (column[2])

For \( i = 1 \ldots M \)
If \( i > 1 \), then:
- Free(column[i-2])
- Allocate(column[i])

For \( j = 1 \ldots N \)
\( F(i, j) = \ldots \)

What about the pointers?

Finding the best back-pointer for current column

Now, using 2 columns of space, we can compute for \( k = 1 \ldots M \), \( F(M/2, k), F(M/2, N-k) \)

PLUS the backpointers
Best forward-pointer for current column

- Now, we can find \( k' \) maximizing \( F(M/2, k) + F(M/2, N-k) \)
- Also, we can trace the path exiting column \( M/2 \) from \( k' \)

Recursively find midpoint for left & right

- Iterate this procedure to the left and right!

Total time cost of linear-space alignment

\[
\text{Total Time:} \quad cMN + cMN/2 + cMN/4 + \ldots = 2cMN = O(MN)
\]

Total Space: \( O(N) \) for computation, \( O(N+M) \) to store the optimal alignment

Summary

- Dynamic programming
  - Reuse of computation
  - Order sub-problems. Fill table of sub-problem results
  - Read table instead of repeating work (ex: Fibonacci)
- Sequence alignment
  - Edit distance and scoring functions
  - Dynamic programming matrix
  - Matrix traversal path ⇔ Optimal alignment
- Thursday: Variations on sequence alignment
  - Local and global alignment
  - Affine gap penalties
  - Algorithmic speed-ups
- Recitation:
  - Dynamic programming applications
  - Probabilistic derivations of alignment scores