

## Dynamic Programming 4

Chapter 7, Algorithm Design

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c31>

---

---

---

---

---

---

---

---

### Longest Common Subsequence (LCS)

- ◆ Given two sequences  $x[1..m]$  and  $y[1..n]$ , find the longest subsequence which occurs in both
- ◆ Ex:  $x = \text{"A B C B D A B"}$ ,  $y = \text{"B D C A B A"}$
- ◆ "B C" and "A A" are both subsequences of both  
*What is the LCS?*
- ◆ Brute-force algorithm: For every subsequence of  $x$ , check if it's a subsequence of  $y$ .
  - » *How many subsequences of  $x$  are there?*
  - » *What will be the running time of the brute-force algorithm?*

2

---

---

---

---

---

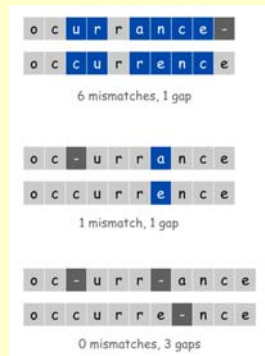
---

---

---

### LCS Can Find Edit Distance

- ◆ How similar are these two strings?
  - » occurrence
  - » occurence



3

---

---

---

---

---

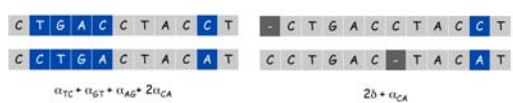
---

---

---

## Application of Edit Distance

- ◆ Automated correction in MS Word
- ◆ Speech recognition
- ◆ Computational biology
  
- ◆ Definition of Edit Distance
  - » Gap penalty  $\delta$
  - » Mismatch penalty  $\alpha_{pq}$
  - » Edit-distance = sum of gap and mismatch penalties.



4

---

---

---

---

---

---

---

---

---

---

---

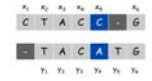
---

## Sequence Alignment

- ◆ Problem: Given two strings  $X = x_1x_2\dots x_m$  and  $Y = y_1y_2\dots y_n$ , find an alignment of minimal cost.
- ◆ Def: An alignment  $M$  is a list of ordered pairs  $x_i-y_j$  such that each item occurs in at most one pair and the  $x_i$ 's and  $y_j$ 's are subsequences of  $X$  and  $Y$ , respectively.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i: x_i \text{ unmatched}} \delta + \sum_{j: y_j \text{ unmatched}} \delta}_{\text{gap}}$$

- ◆ Ex: CTACCG vs TACATG



5

---

---

---

---

---

---

---

---

---

---

---

---

## Sequence Alignment: Problem Structure

- ◆ Def. Let  $OPT(i, j)$  be the minimal cost of aligning strings  $x_1\dots x_i$  and  $y_1\dots y_j$ .
  - » Pay mismatch for  $x_i-y_j$  plus  $OPT(i-1, j-1)$
  - » Pay penalty for unmatched  $x_i$  plus  $OPT(i-1, j)$
  - » Pay penalty for unmatched  $y_j$  plus  $OPT(i, j-1)$ .

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

6

---

---

---

---

---

---

---

---

---

---

---

---

## Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {  
  for i = 0 to m  
    M[0, i] = iδ  
  for j = 0 to n  
    M[j, 0] = jδ  
  
  for i = 1 to m  
    for j = 1 to n  
      M[i, j] = min(α[xi, yj] + M[i-1, j-1],  
                  δ + M[i-1, j],  
                  δ + M[i, j-1])  
  
  return M[m, n]  
}
```

- ♦ Analysis: O(mn) time and space

7

---

---

---

---

---

---

---

---

## Linear Space Algorithm

```
Linear-space-alignment-value(m, x, X, Y, δ, α)  
  for j = 0 to n M[j] = jδ  
  for i = 1 to m  
    corner = M[0]  
    M[0] = left = M[0] + δ  
    for j = 1 to n  
      up = M[j]  
      x = min((α[X[i], Y[j]] + corner), (δ + up), (δ + left))  
      M[j] = left = x  
      corner = up  
  return M[n]
```

corner	up
left	x

Optimal value can be computed in O(mn) time and O(m+n) space  
How about the actual alignment?

8

---

---

---

---

---

---

---

---

## Linear Space Algorithm

- ♦ Theorem [Hirschberg 1975] Optimal alignment can be computed in O(m+n) space and O(mn) time.
- ♦ Clever combination of divide-and-conquer and dynamic programming.

9

---

---

---

---

---

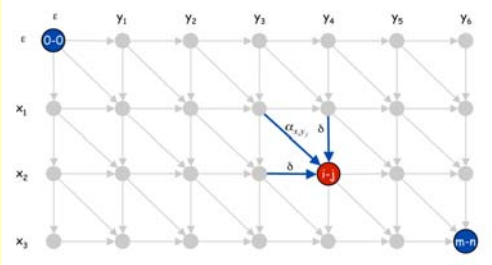
---

---

---

## Linear Space Algorithm

- ◆ Edit distance graph: Let  $f(i,j)$  be the shortest distance from  $(0,0)$  to  $(i,j)$ .
- ◆ Observation:  $f(i,j) = \text{OPT}(i,j)$ .



10

---

---

---

---

---

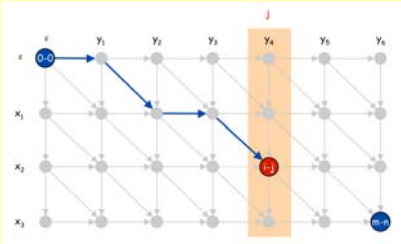
---

---

---

## Linear Space Algorithm

- ◆ Edit distance graph: Let  $f(i,j)$  be the shortest distance from  $(0,0)$  to  $(i,j)$ .
- ◆ Observation: can compute  $f(i,j)$  for all  $i$  and any fixed  $j$  in  $O(m+n)$  space and  $O(m,n)$  time.



11

---

---

---

---

---

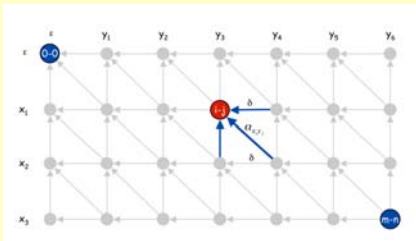
---

---

---

## Linear Space Algorithm

- ◆ Edit distance graph: Let  $g(i,j)$  be the shortest distance from  $(i,j)$  to  $(m,n)$ .
- ◆ Observation: Can compute  $g(i,j)$  by reversing the edges and inverting the roles of  $(0,0)$  and  $(m,n)$ .



12

---

---

---

---

---

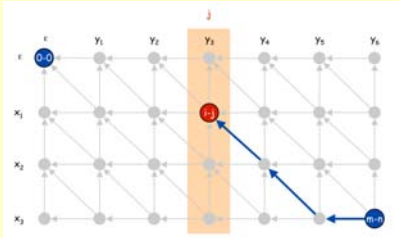
---

---

---

## Linear Space Algorithm

- ◆ Edit distance graph: Let  $g(i,j)$  be the shortest distance from  $(i,j)$  to  $(m,n)$ .
- ◆ Observation:  $g(i,j)$  can be computed for all  $i$  and fixed  $j$  in  $O(m+n)$  space and  $O(mn)$  time.



13

---

---

---

---

---

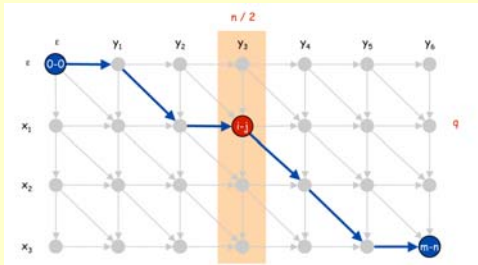
---

---

---

## Linear Space Algorithm

- ◆ Edit distance graph: Let  $f(i,j)$  and  $g(i,j)$  be the shortest distance from  $(0,0)$  to  $(i,j)$  and  $(i,j)$  to  $(m,n)$ , resp.
- ◆ Observation:  $\text{OPT}(m,n) = f(i,n/2) + g(i,n/2)$  for some  $i$ .



14

---

---

---

---

---

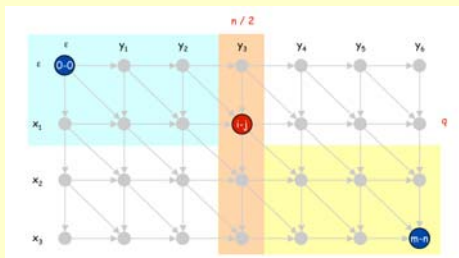
---

---

---

## Linear Space Algorithm

- ◆ Divide: find index  $q$  that minimizes  $f(q, n/2) + g(q, n/2)$  using DP.
- ◆ Align  $x_q$  and  $y_{n/2}$
- ◆ Conquer: recursively compute optimal alignment in each region.



15

---

---

---

---

---

---

---

---

## Linear Space Algorithm

- ♦ Divide: find index  $q$  that minimizes  $f(q, n/2) + g(q, n/2)$  using DP.
- ♦ Align  $x_q$  and  $y_{n/2}$
- ♦ Conquer: recursively compute optimal alignment in each region.

```
Divide-Conquer-Alignment(s, m, t, n, X, Y,  $\delta$ ,  $\alpha$ )
  if  $m-s \leq 2$  or  $n-t \leq 2$  compute it using old DP.
  Call Alignment-value(s, m, t, (t+n+1)/2, X, Y,  $\delta$ ,  $\alpha$ )
  Call Backward-Alignment-value(s, m, (t+n+3)/2, n, X, Y,  $\delta$ ,  $\alpha$ )
  Let  $q$  be the index minimizing  $f(q, n/2)+g(q, n/2)$ 
  Add  $(q, (t+n+1)/2)$  to global list M
  Divide-Conquer-Alignment(s, q, t, (t+n+1)/2, X, Y,  $\delta$ ,  $\alpha$ )
  Divide-Conquer-Alignment(q+1, m, (t+n+3)/2, n, X, Y,  $\delta$ ,  $\alpha$ )
  return M
```

16 What's the complexity of this algorithm?

---

---

---

---

---

---

---

---