COT 6405
ANALYSIS OF ALGORITHMS

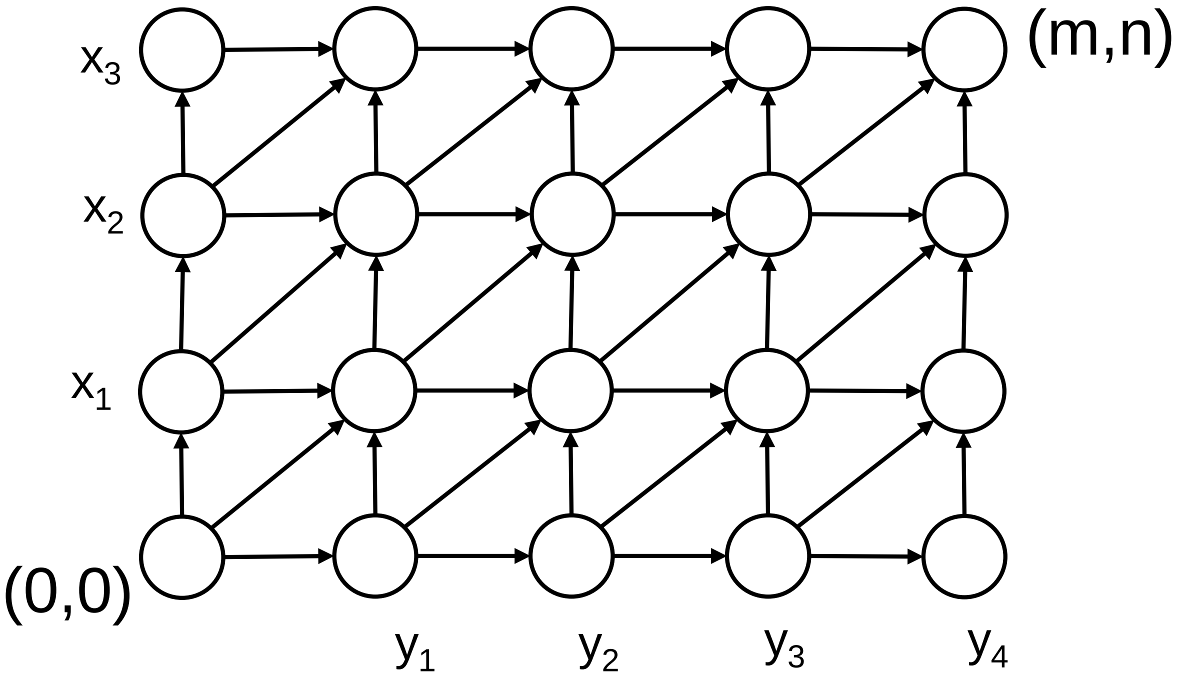
Sequence Alignment in Linear Space via Divide-and-Conquer

Computer & Electrical Engineering and Computer Science Department
Florida Atlantic University

Motivation

- Computing the sequence alignment of two sequences X and Y using the dynamic programming algorithm Alignment(X,Y) has RT = O(mn) and space O(mn)
- This is too large for biological applications where strings are very long
  - if the two strings have ~ 100,000 symbols each, then RT ~ 10 billion primitive operations and space is ~10 billion array
- Objective: enhancement of the sequence alignment algorithm that has RT = O(mn) and space O(m+n)
  - Uses a divide-and-conquer algorithm

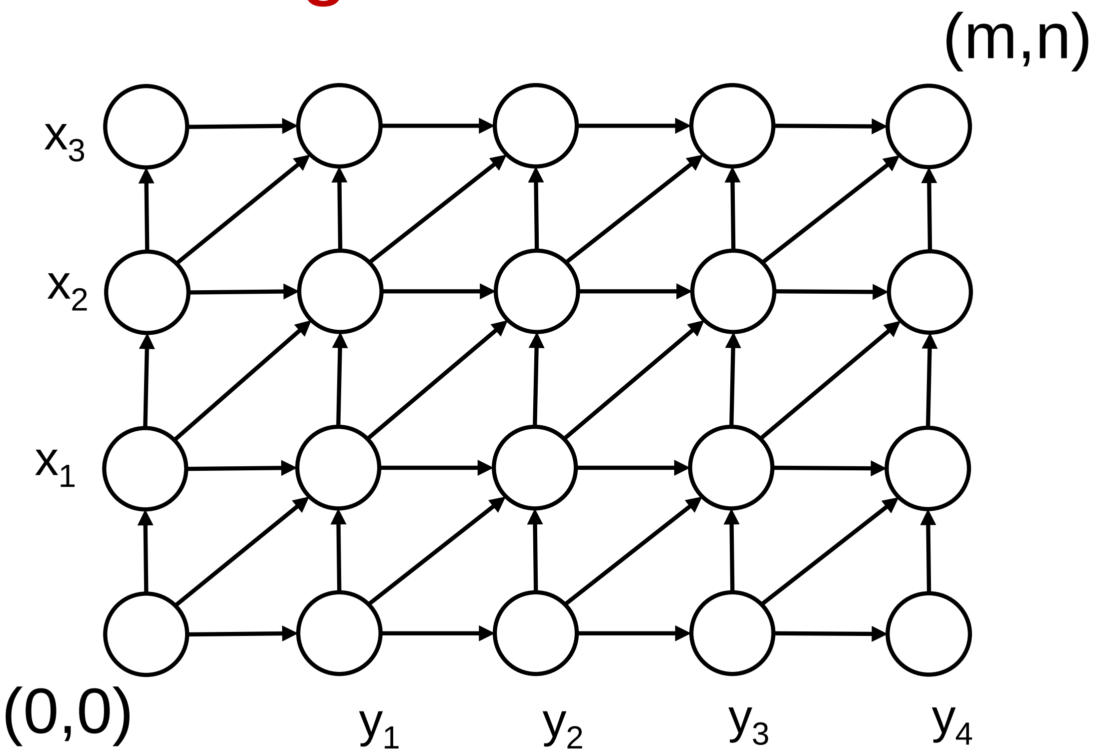
Graph representation of the sequence alignment



- Cost of the edges:
  - horizontal & vertical edges have cost δ
  - diagonal edge (i-1,j-1) to (i,j) has cost $\alpha_{x_i y_j}$
- Value of an optimal alignment is the minimum-cost of a path from (0,0) to (m,n)

Graph representation of the sequence alignment

- Let f(i,j) denote the minimum cost of a path from (0,0) to (i,j) in G_{XY}. Then for all i,j, f(i,j) = OPT(i,j), where OPT(i,j) is the minimum cost of an alignment of X_i and Y_j.



- Finding the optimal alignment is equivalent to constructing the graph G_{XY} with (m+1)(n+1) nodes laid out in a grid and computing the cheapest path between opposite corners
- RT for this approach is O(mn) and space O(mn)

Space-efficient Alignment

- First, we'll show that if we care only about the *value* of an optimal alignment, then it's easy to have linear space
- Key observation: to fill out the array A, we only need information on the current column of A and the previous column of A
- Instead of using array A of size $(m+1)\times(n+1)$, use array B of size $(m+1)\times 2$
- As the algorithm iterates through values of j, entries B[i,0] will hold the *previous* column's value A[i,j-1] and entries of the form B[i,1] will hold the *current* column's values A[i,j]

Designing the Algorithm

Space-Efficient-Alignment (X,Y)

```
array B[0..m,0..1]
initialize B[i,0] = iδ for each i (just as column 0 of A)
for j = 1,…,n
    B[0,j] = jδ (since this corresponds to entry A[0,j])
    for i = 1,…,m
        B[i,1] = min{α_{x_i y_j} + B[i-1,0], δ + B[i-1,1], δ + B[i,0]}
    endfor
    // move col 1 of B to col 0 to make room for the next iteration
    update B[i,0] = B[i,1] for each i
endfor
```

- RT = O(mn)
- space = O(m)

Space-efficient Alignment

- when the algorithm terminates, B[i,1] holds the value of OPT(i,n) for i = 0,…m
  - OPT(m,n) – minimum cost of an alignment of X and Y
- issue: how can we determine the assignment itself?
  - we haven't left enough information to find the alignment
  - B has only the last two columns, so we cannot trace back the optimal alignment (shortest path)
- we need a different approach if we want to recover the optimal alignment

A backward formulation of the DP

- $f(i,j)$ – length of the shortest path $(0,0)$ to (i,j) in the graph G_{XY}

$$f(i,j) = OPT(i,j)$$

- define $g(i,j)$ – length of the shortest path from (i,j) to (m,n) in G_{XY}
- build g using DP in reverse: start with $g(m,n) = 0$, and the answer we want is $g(0,0)$
- for $i < m$ and $j < n$ we have:

$$g(i,j) = \min [\alpha_{x_{i+1}\, y_{j+1}} + g(i+1, j+1), \delta + g(i, j+1), \delta + g(i+1, j)]$$

- g is built using DP backward from (m,n)
- we can also design the space-efficient version,

Backward-Space-Efficient-Alignment(X,Y)

in space $O(m)$ and RT = $O(mn)$

Combining the Forward and Backward Formulations

- The length of the shortest corner-to-corner path in G_{XY} that passes through (i,j) is $f(i,j) + g(i,j)$
- Let k be any number in $\{0,\ldots,n\}$ and let q be an index minimizes the quantity $f(q,k) + g(q,k)$. Then there is a corner-to-corner path of minimum length that passes through the node (q,k).

Designing the divide-and-conquer algorithm

- divide G_{XY} along the center column and compute f(i,n/2) and g(i,n/2) for each i, using the two space-efficient algorithms
- find the minimum f(i,n/2) + g(i,n/2) for some value i
- then there is a shortest corner-to-corner path that passes through (i,n/2)
- recursively find the shortest-path in G_{XY} between (0,0) and (i,n/2) and in the portion between (i,n/2) and (m,n)
- MAIN IDEA:
  - Apply these recursive calls sequentially and reuse the working space from one call to the next
- then the space usage is O(m+n)

Designing the divide-and-conquer algorithm

- maintain a globally accessible list P with nodes on the shortest corner-to-corner path as they are discovered
  - initially, P is empty
  - P has at most m+n entries, since any path has at most m+n edges
- notation:

  X[i:j], for $1 \le i \le j \le m$, is the substring $x_i x_{i+1} \ldots x_j$

  similar for Y[i:j]
- assume for simplicity that n is a power of 2

Designing the divide-and-conquer algorithm

Divide-and-Conquer-Alignment(X,Y)

m is the number of symbols in X

n is the number of symbols in Y

if m $\leq$ 2 or n $\leq$ 2 then

  compute optimal alignment using Alignment(X,Y)

call Space-Efficient-Alignment(X,Y[1:n/2])

call Backward-Space-Efficient-Alignment(X,Y[n/2+1:n])

let q be the index minimizing f(q,n/2) + g(q,n/2)

add (q,n/2) to the global list P

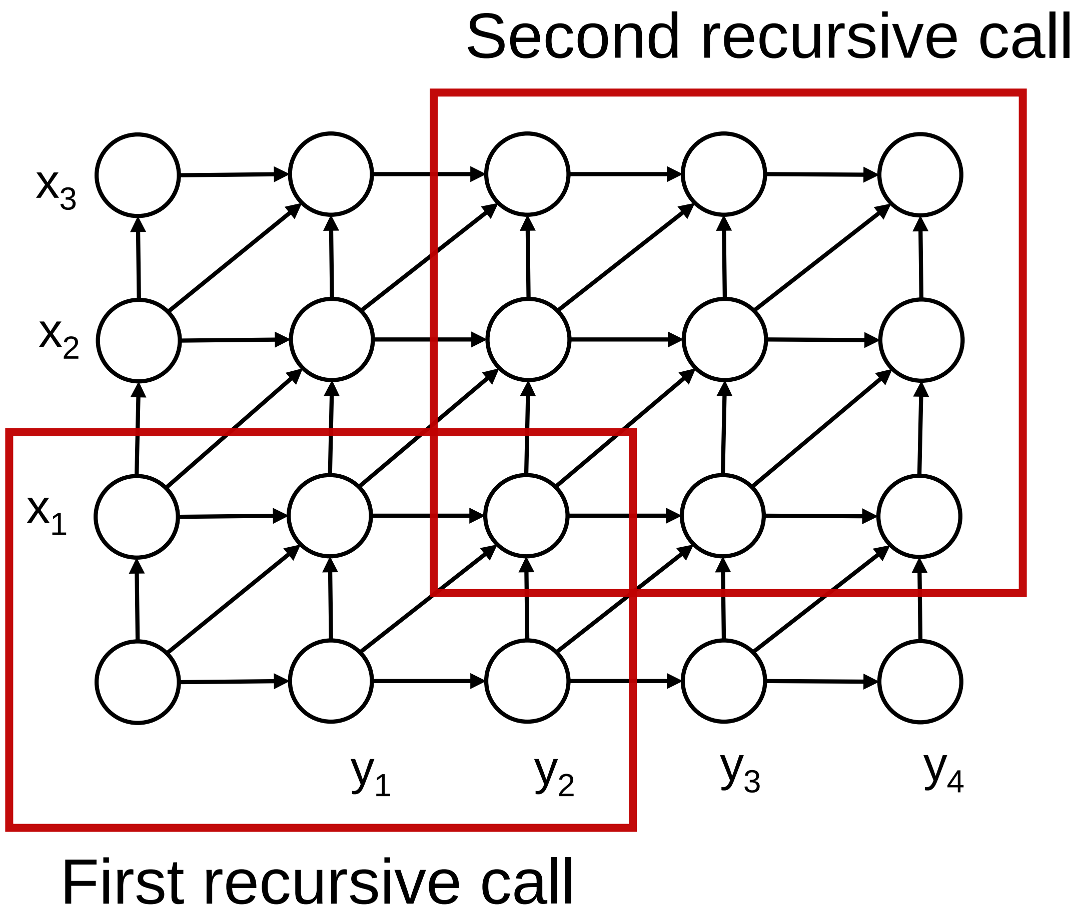
Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])

Divide-and-Conquer-Alignment(X[q+1:m],Y[n/2+1:n])

return P

✓ Space used is O(m + n)

Example


Second recursive call
x_3
x_2
x_1
y_1
y_2
y_3
y_4
First recursive call

RT analysis

The RT of Divide-and-Conquer-Alignment on strings of length m and n is $O(mn)$.

Proof:

$T(m,n)$ – running time

$T(m,n) \leq cmn + T(q,n/2) + T(m-q,n/2)$

$T(m,2) \leq cm$

$T(2,n) \leq cn$

Particular case: $m = n$ and q is in the middle

$T(n) \leq cn^2 + 2T(n/2)$

case 3 of the Master Theorem $\Rightarrow T(n) = \Theta(n^2)$

RT analysis

General case:

$$T(m,n) \le cmn + T(q,n/2) + T(m-q,n/2)$$

Show by induction that $T(m,n) = O(mn)$, that means

$$T(m,n) \le kmn \text{ for some constant } k$$

Base case: $m \le 2$ or $n \le 2$ is true

Inductive step:

$$\begin{aligned} T(m,n) &\le cmn + T(q,n/2) + T(m-q,n/2) \\ &\le cmn + kqn/2 + k(m-q)n/2 \\ &= cmn + kqn/2 + kmn/2 - kqn/2 \\ &= (c + k/2)mn \end{aligned}$$

Inductive step works for $c + k/2 = k \Rightarrow c = k/2$ or $k = 2c$