A Scalable Approximation Algorithm for Weighted Longest Common Subsequence

Jeremy Buhler, **Thomas Lavastida**,
Kefu Lu, Benjamin Moseley

Carnegie Mellon University
Tepper School of Business

Longest Common Subsequence

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Value = number of pairs (i, j) with $x[i] = y[j]$

$\uparrow$
y

G						
G						
A						
T						
C						
	A	C	T	G	G	A

$x \rightarrow$

Longest Common Subsequence

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Value = number of pairs (i, j) with $x[i] = y[j]$

$\uparrow$
y

G					4	
G	3					
A						
T				2		
C			1			
	A	C	T	G	G	C

$x \rightarrow$

Invalid correspondence

Longest Common Subsequence

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Value = number of pairs (i, j) with $x[i] = y[j]$

$\uparrow$ y

G					4	
G				3		
A						
T			2			
C		1				
	A	C	T	G	G	C

$x \rightarrow$

Valid correspondence
Value = 4

Weighted Longest Common Subsequence

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Weight function $f: \Sigma \times \Sigma \to \mathbb{N}$
- Value = $\sum_k f(x[i_k], y[j_k])$

G					4	
G				3		
A						
T			2			
C		1				
	A	C	T	G	G	C

$\uparrow$ y

$x \rightarrow$

G	0	0	1	2
C	0	0	1	1
T	1	1	0	0
A	3	1	0	0
	A	T	C	G

Value = 6

Weighted Longest Common Subsequence

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Weight function $f: \Sigma \times \Sigma \rightarrow \mathbb{N}$
- Value = $\sum_k f(x[i_k], y[j_k])$

↑ y

G					3	
G				2		
A	1					
T						
C						
	A	C	T	G	G	C

$x \rightarrow$

G	0	0	1	2
C	0	0	1	1
T	1	1	0	0
A	3	1	0	0
	A	T	C	G

Value = 7

All-Substrings WLCS

- Strings x, y over alphabet Σ, $|x| = n \geq m = |y|$
- Correspondence = monotone sequence of index pairs $\{(i_k, j_k)\}_{k=1}^{\ell}$
- Weight function $f: \Sigma \times \Sigma \to \mathbb{N}$
- Value = $\sum_k f(x[i_k], y[j_k])$
- Goal: Compute matrix C where $C(i, j)$ = WLCS value between x and $y[i:j]$

WLCS and AWLCS

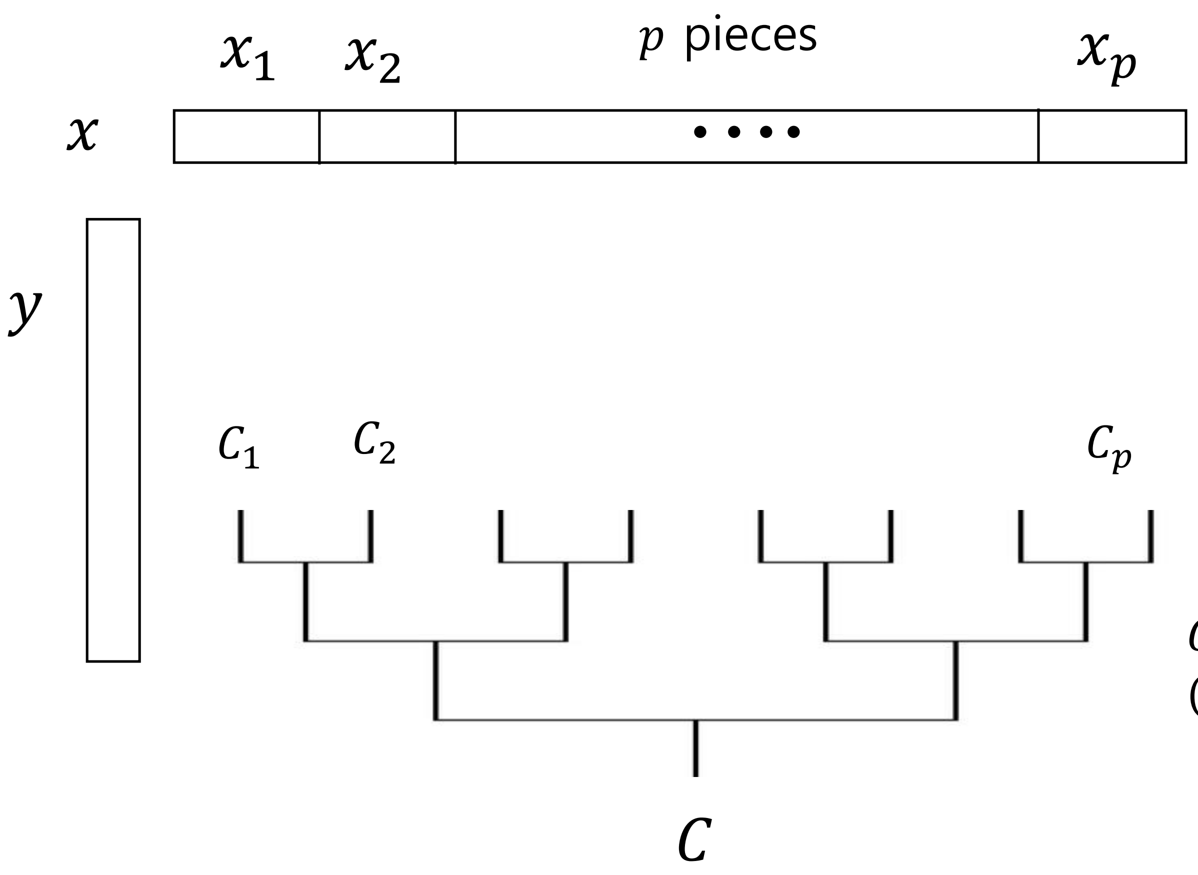
- Applicable to bioinformatics problems
- C matrix from AWLCS useful for inferring structure in the strings
  - Approximate tandem repeats
  - Circular alignments
- WLCS solvable by a dynamic programming algorithm in $O(nm)$ time
- AWLCS solvable in time $O(nm \log m)$ [Schmidt 1998]
- Unweighted case of AWLCS solvable in $O(nm)$ time [Alves et al. 2008]

But what about parallel algorithms?

Parallel Algorithms for WLCS

- Standard dynamic programs do not parallelize easy
  - Dependency chains of length $\Theta(n+m)$
- Divide-and-conquer approaches
  - Strings x_1, x_2, y
  - C_1, C_2 AWLCS matrices for strings (x_1, y) and (x_2, y)
  - Matrix multiplication over the ring $(\max, +)$
  - $C_1 \cdot C_2$ is the AWLCS matrix for $(x_1 \cdot x_2, y)$

Divide-and-Conquer Approaches



$O(\log p)$ combining steps via $(\max, +)$ matrix multiplication

Divide-and-Conquer Approaches

- $A(m) =$ time to multiply two $m \times m$ AWLCS matrices
- $B(n, m)$ = time to compute AWLCS on strings of length n, m
- With p processors, using divide-and-conquer gets a running time of:

$$B\left(\frac{n}{p}, m\right) + A(m) \log p$$

Divide-and-Conquer Approaches

- Need to make the matrix multiplication step fast
- Exploit the **Monge property** of AWLCS matrices

$$C(i,j) + C(k,\ell) \le C(i,\ell) + C(k,j)$$

- For unweighted case, [Tiskin 2015] gives $A(m) = O(m \log m)$
- For AWLCS, $A(m) = O(m^2)$ [Russo 2012]

Our Results (Divide-and-Conquer)

Theorem 1: For any $\epsilon > 0$, there is a parallel algorithm with

$$O\left(B\left(\frac{n}{p}, m\right) + \frac{1}{\epsilon^2} m \log^2 W \log^2 n \log p\right)$$

running time which finds a $(1 - \epsilon)$-approximate WLCS solution.

- p = number of processors
- W = maximum weight of a correspondence

Our Results (Base Case)

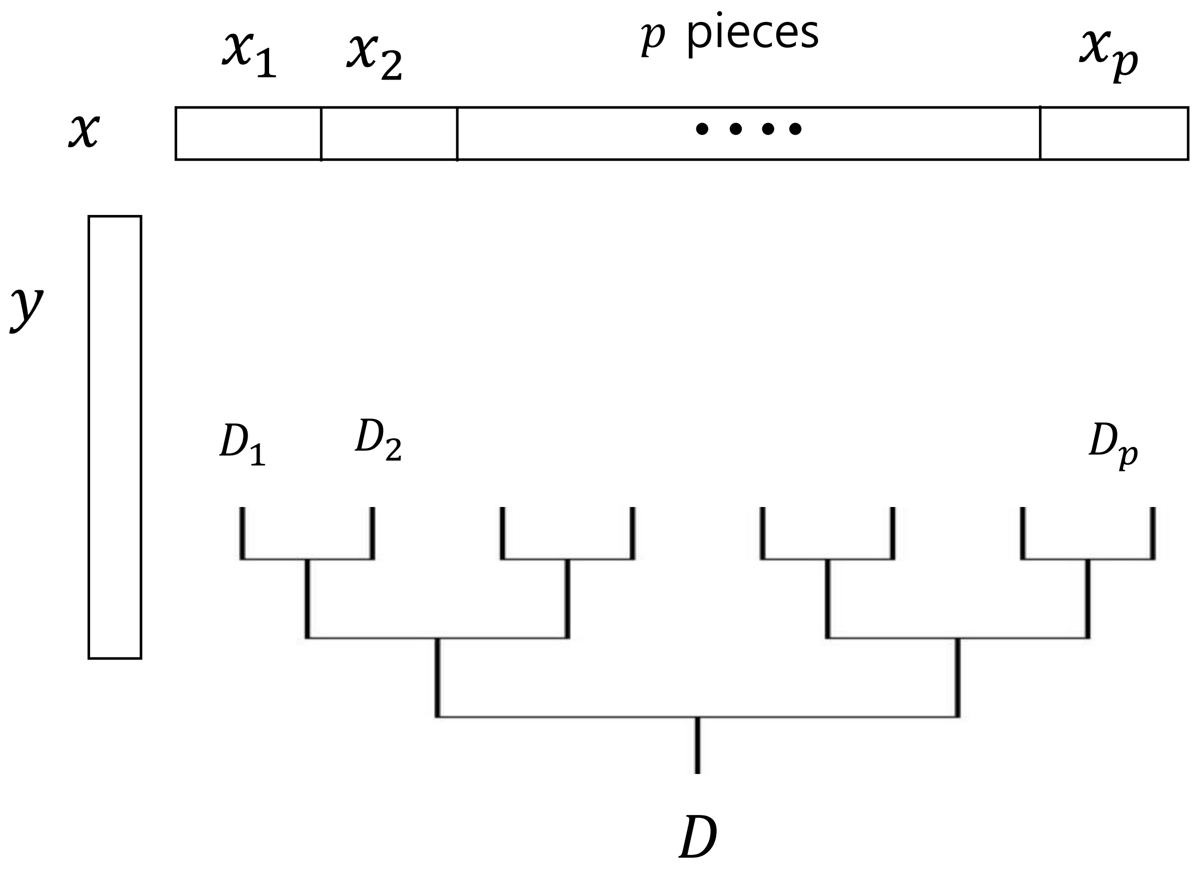
- Look at case where f is bounded by σ
- Extend the result of [Alves et al. 2008] to this case

Theorem 2: There is a sequential algorithm with $O(\sigma n m)$ running time which finds an implicit representation of the AWLCS matrix.

- Plugging into Theorem 1 gives overall running time

$$O\left(\frac{mn\sigma}{p} + \frac{1}{\epsilon^2} m \log^2 \sigma m \log^2 n \log p\right)$$

Algorithm Sketch (Divide-and-Conquer)



$O(\log p)$ combining steps via "sketched" dynamic program

“Sketched” Dynamic Program

- Recall $C(i,j) =$ WLCS value between x and $y[i:j]$
- Define $D(i,w) =$ smallest index j s.t. $C(i,j) \geq w$
- Computing for all w is intractable
- Instead look at powers of $\approx 1 + \frac{\epsilon}{\log n}$
- No longer exact, but $(1-\epsilon)$-approximate

Algorithm Sketch (Base Case)

- Generalization of [Alves et al. 2008] to weighted case
- Two sets of indices - h and v indices
  - Track where increments occur in the AWLCS matrix
  - Existence follows from the Monge property
- Give a recurrence to compute these, naïvely in time $O(\sigma^2 nm)$
- A careful algorithm computes them in time $O(\sigma nm)$

Conclusion

- Study WLCS and AWLCS
- Parallel WLCS algorithm w/ $O\left(\frac{mn\sigma}{p} + \frac{1}{\epsilon^2} m \log^2 \sigma m \log^2 n \log p\right)$ running time
- Sequential AWLCS algorithm w/ $O(mn\sigma)$ running time

Thank you!