Alignment of Long Sequences: LAGAN

BMI/CS 776

www.biostat.wisc.edu/bmi776/

Spring 2019

Colin Dewey

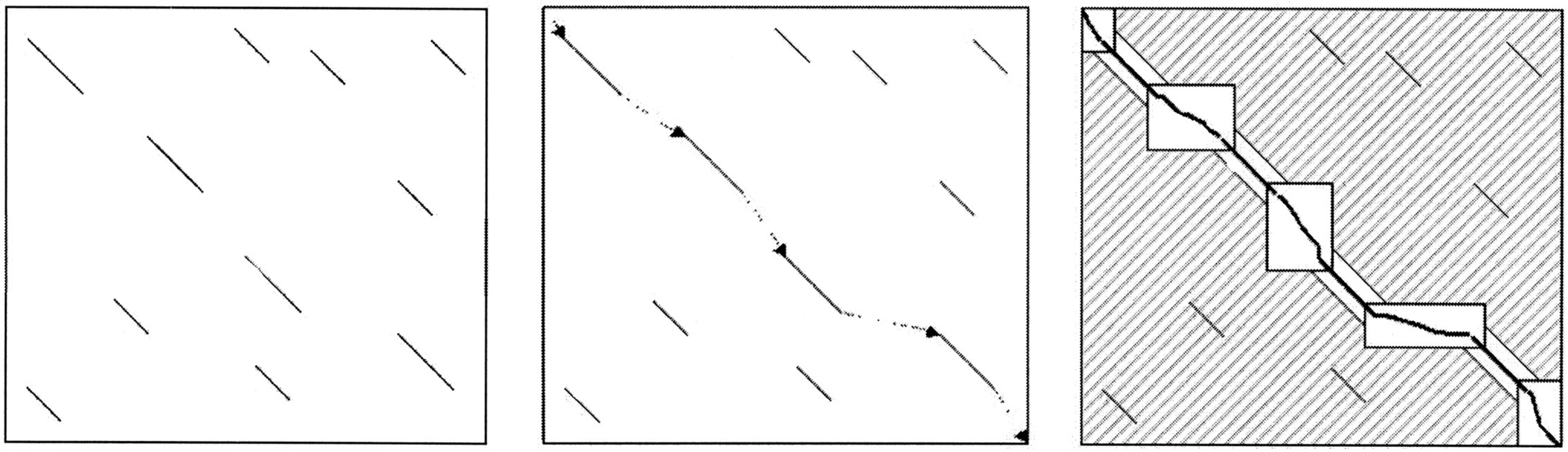
colin.dewey@wisc.edu

Goals for Lecture

Key concepts

- using tries and threaded tries to find alignment seeds
- using sparse dynamic programming (DP) to find a chain of local alignments
- constrained dynamic programming to align between/around anchors

LAGAN: Three Main Steps



Brudno et al. *Genome Research*, 2003

General

1. Pattern matching to find seeds for global alignment
2. Find a good chain of anchors
3. Fill in with standard but constrained alignment

LAGAN

1. Threaded tries to obtain seeds
2. Sparse dynamic programming for chaining
3. Dynamic programming for gap filling

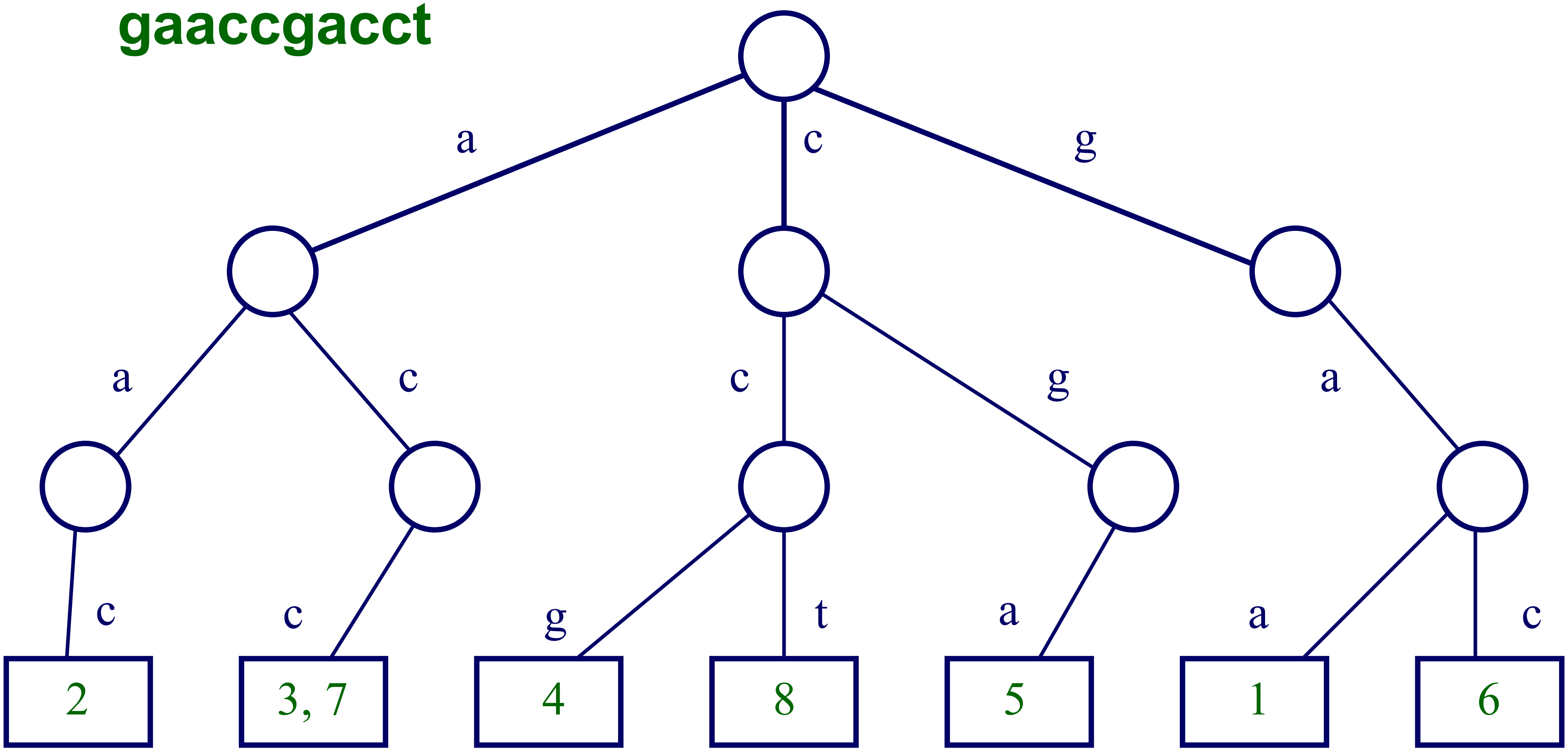
Step 1: Finding Seeds in LAGAN

- *Degenerate k-mers*: matching *k*-long sequences with a small number of mismatches allowed
- By default, LAGAN uses 10-mers and allows 1 mismatch

cacg	cgcgctacat	acct
acta	cgcggtacat	cgta

Finding Seeds in LAGAN

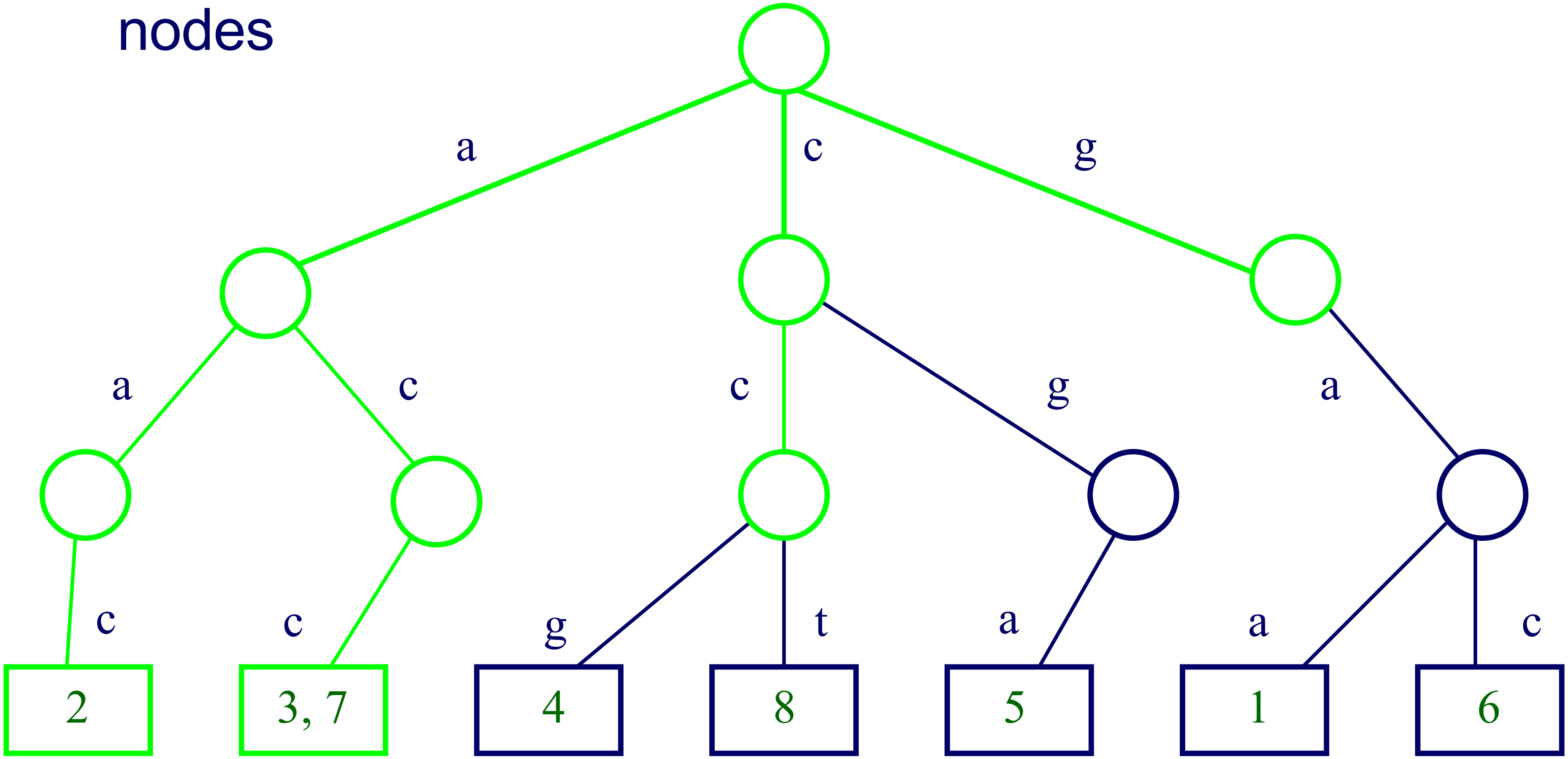
- Example: a *trie* to represent all 3-mers of the sequence **gaaccgacct**



- One sequence is used to build the trie
- The other sequence (the query) is "walked" through to find matching *k*-mers

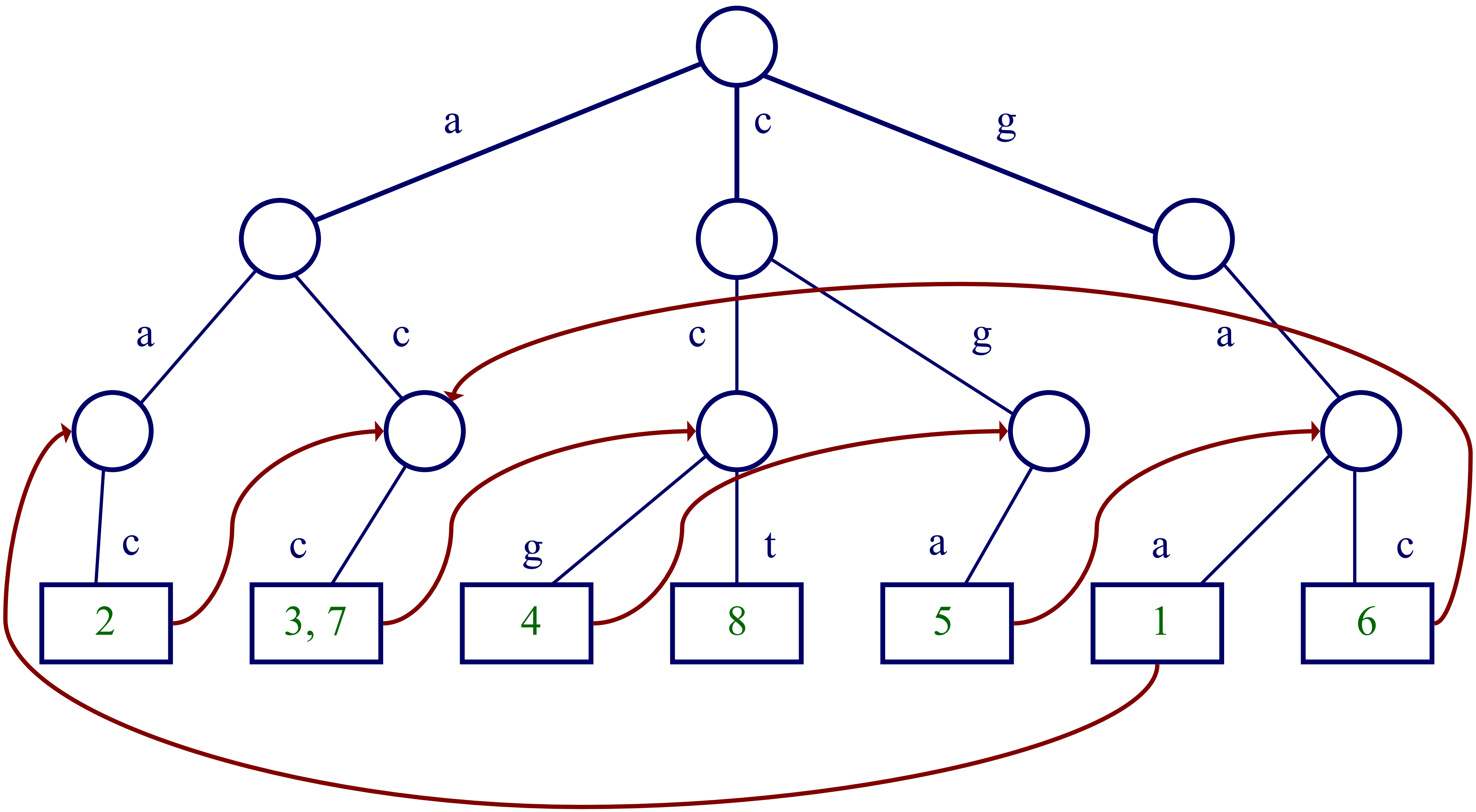
Allowing Degenerate Matches

- Suppose we're allowing 1 base to mismatch in looking for matches to the 3-mer **acc**; need to explore green nodes



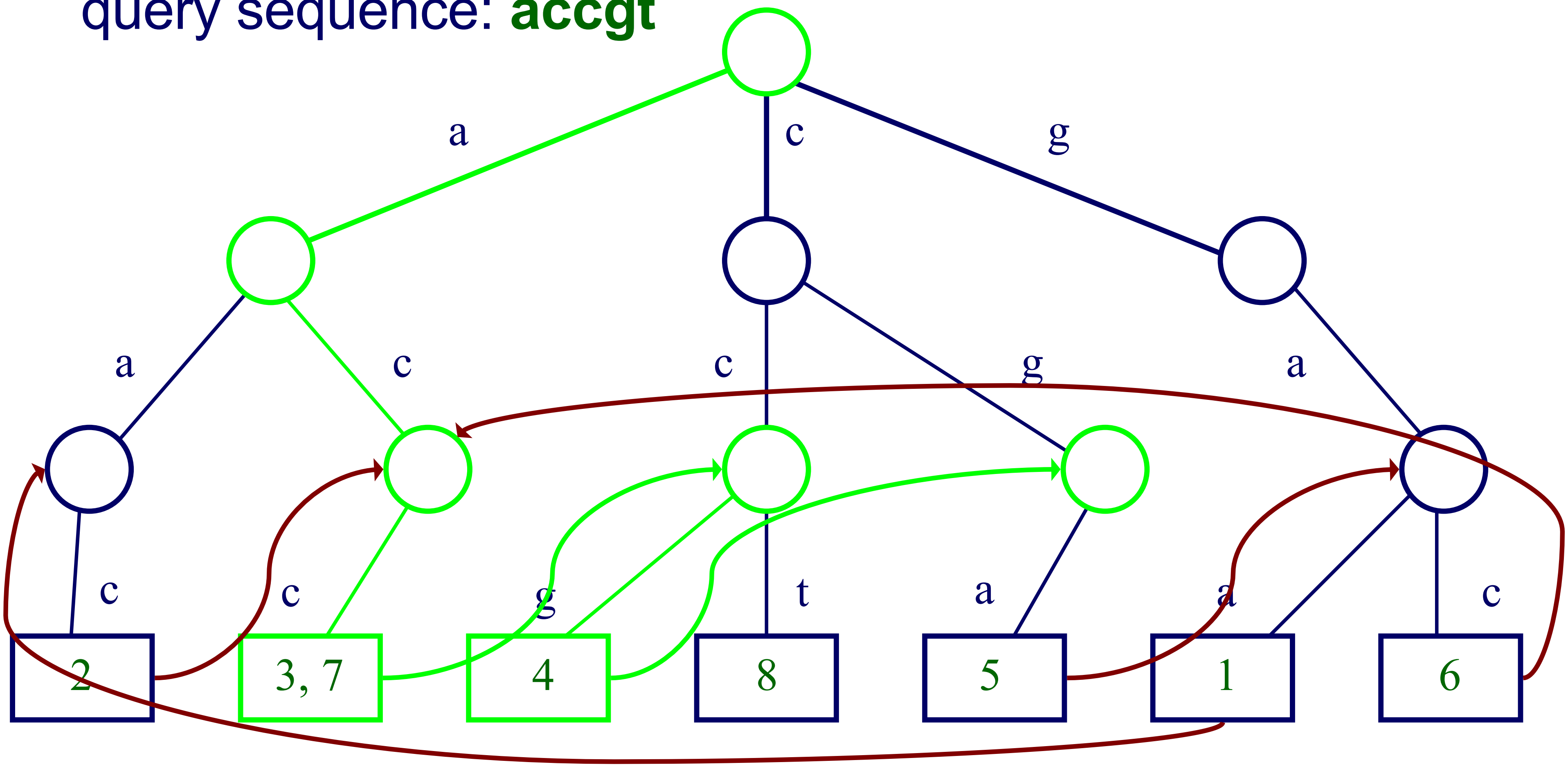
LAGAN Uses Threaded Tries

- In a *threaded trie,* each leaf for word $w_1 \ldots w_k$ has a back pointer to the node for $w_2 \ldots w_k$



Traversing a Threaded Trie

- Consider traversing the trie to find 3-mer matches for the query sequence: **accgt**



- Usually requires following only two pointers to match against the next *k*-mer, instead of traversing tree from root for each

Step 1b: Chaining Seeds in LAGAN

- can chain seeds s_1 and s_2 if
  - the indices of s_1 > indices of s_2 (for both sequences)
  - s_1 and s_2 are near each other
- keep track of seeds in the "search box" as the query sequence is processed

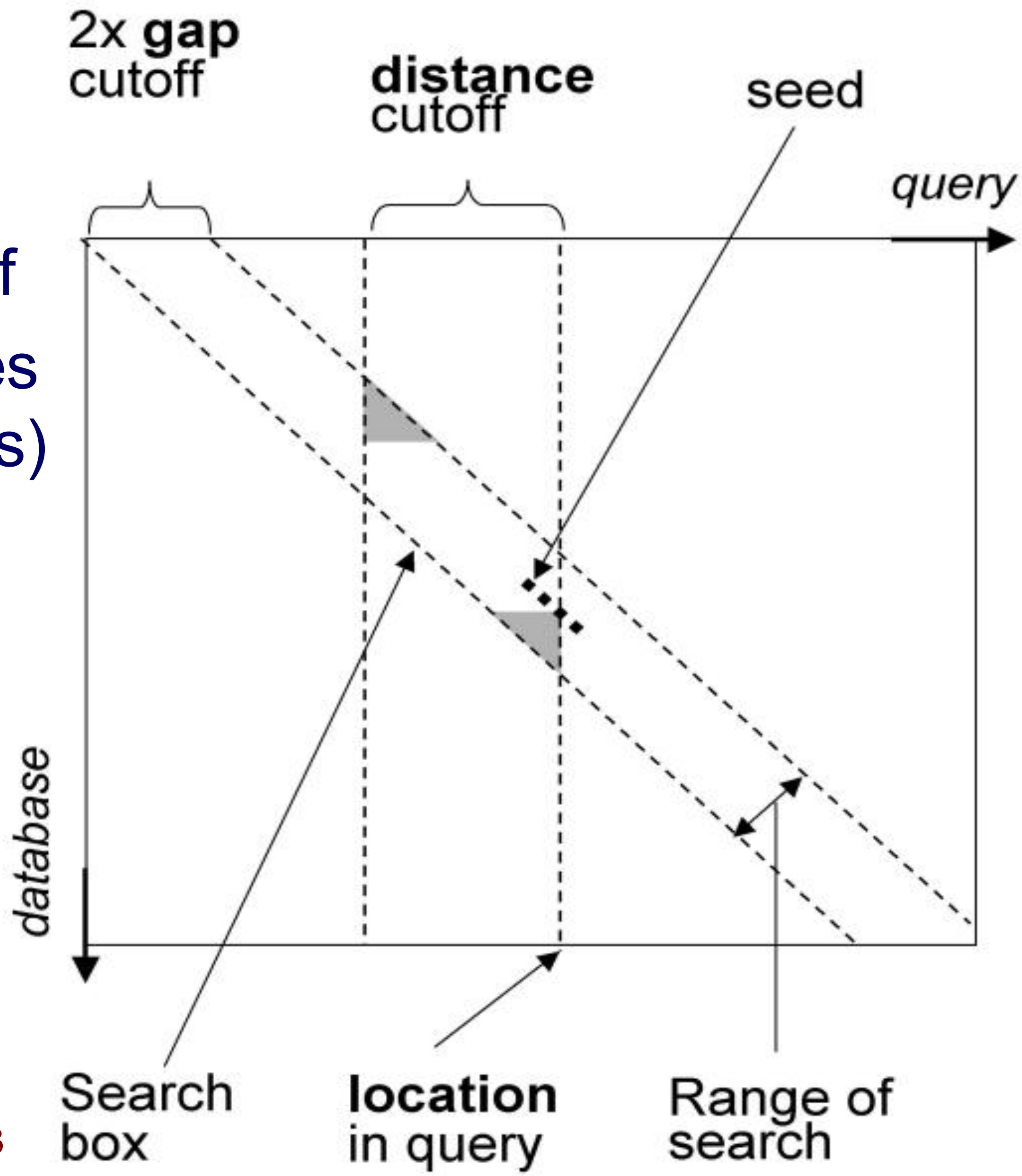
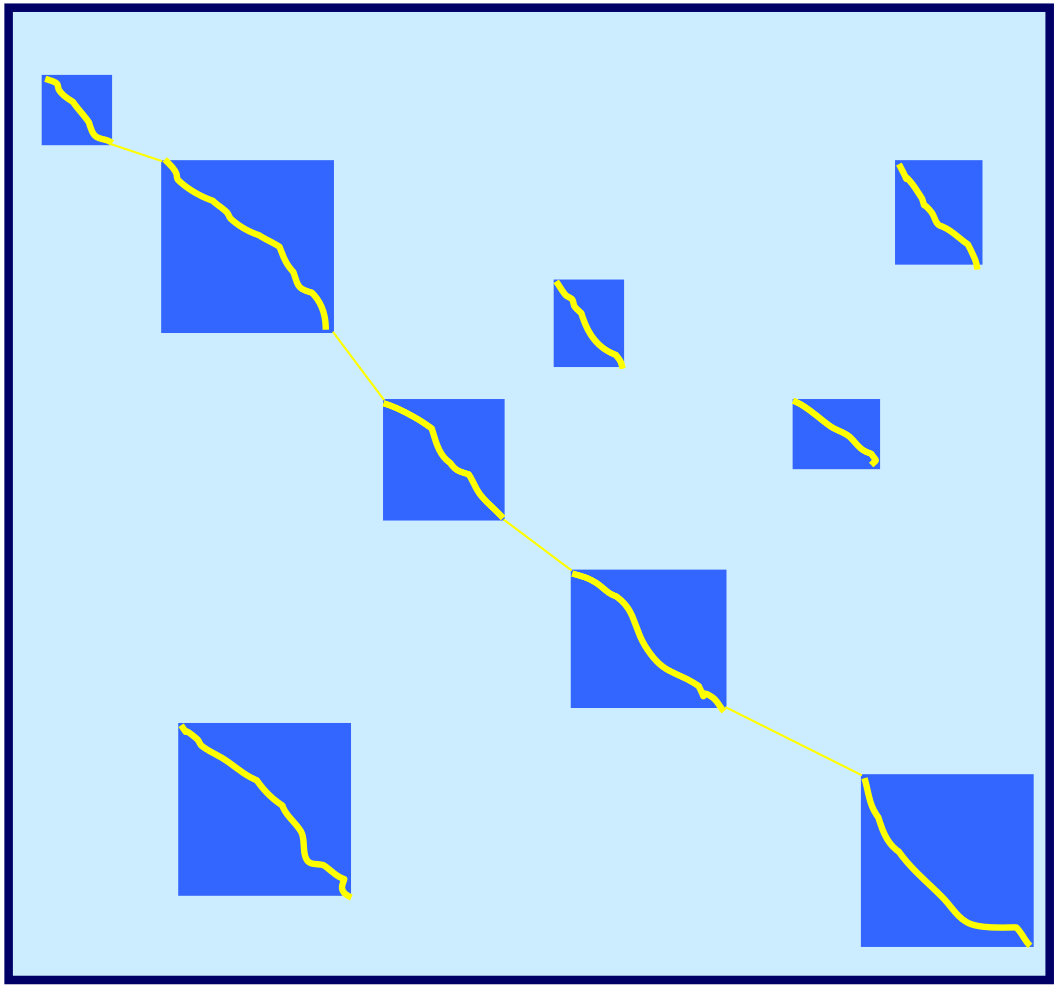


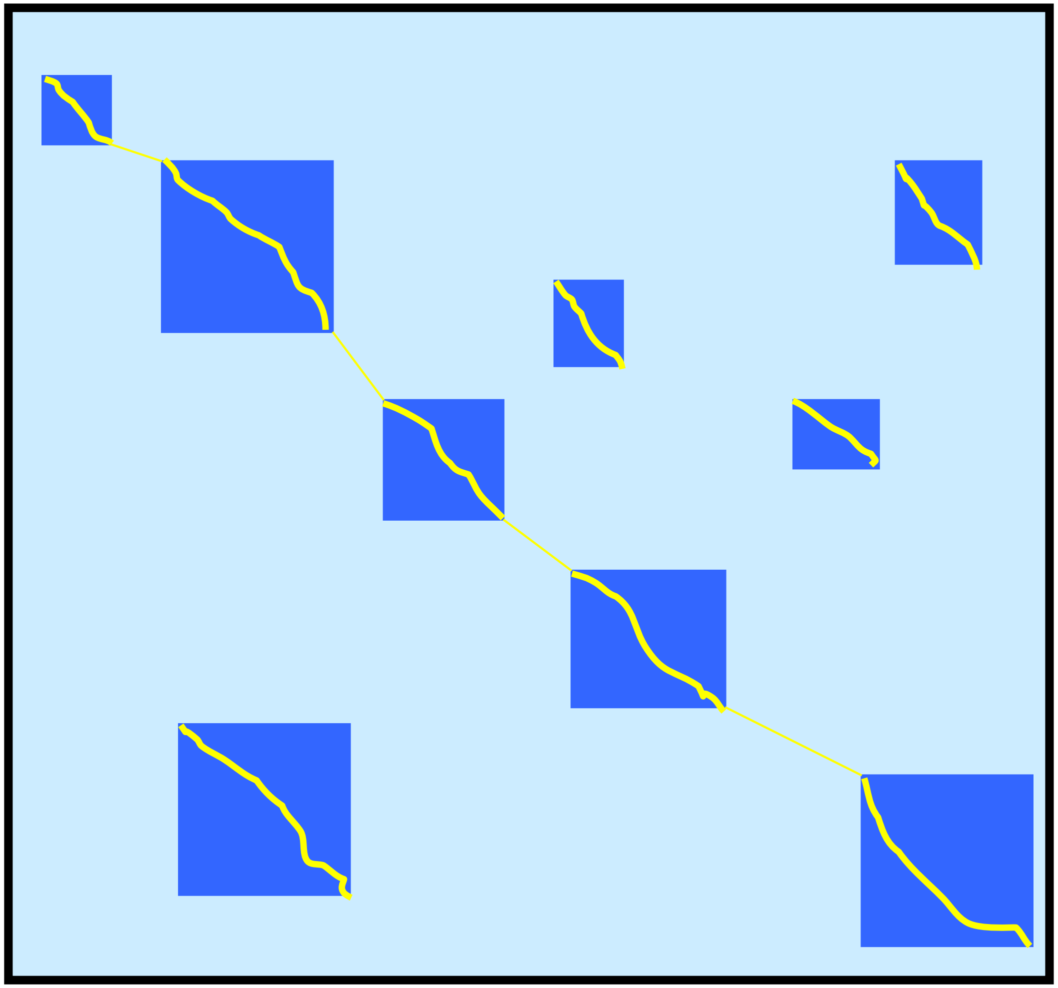
Figure from: Brudno et al. *BMC Bioinformatics*, 2003

Step 2: Chaining in LAGAN

- use *sparse dynamic programming* to chain local alignments



The Problem: Find a Chain of Local Alignments



$(x,y) \rightarrow (x',y')$

requires

$x < x'$
$y < y'$

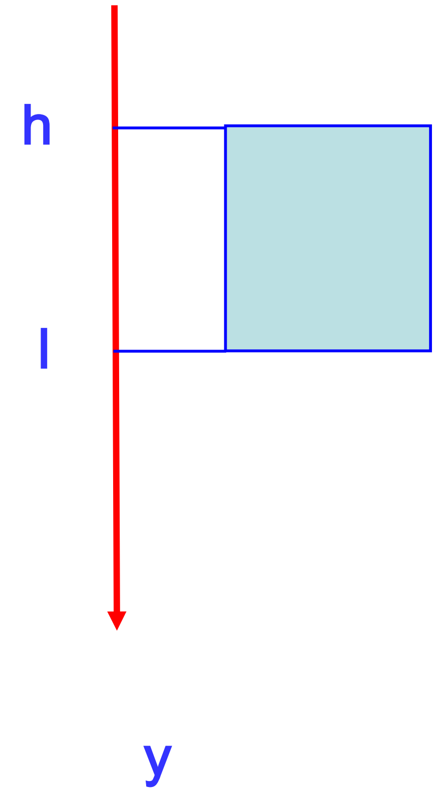
Each local alignment has a weight

FIND the chain with highest total weight

Slide from Serafim Batzoglou, Stanford University

Sparse DP for rectangle chaining

- 1,…, N: rectangles
- (h_j, l_j): y-coordinates of rectangle j
- w(j): weight of rectangle j
- V(j): optimal score of chain ending in j
- L: list of triplets $(l_j, V(j), j)$
  - L is sorted by l_j: smallest (North) to largest (South) value
  - L is implemented as a balanced binary tree

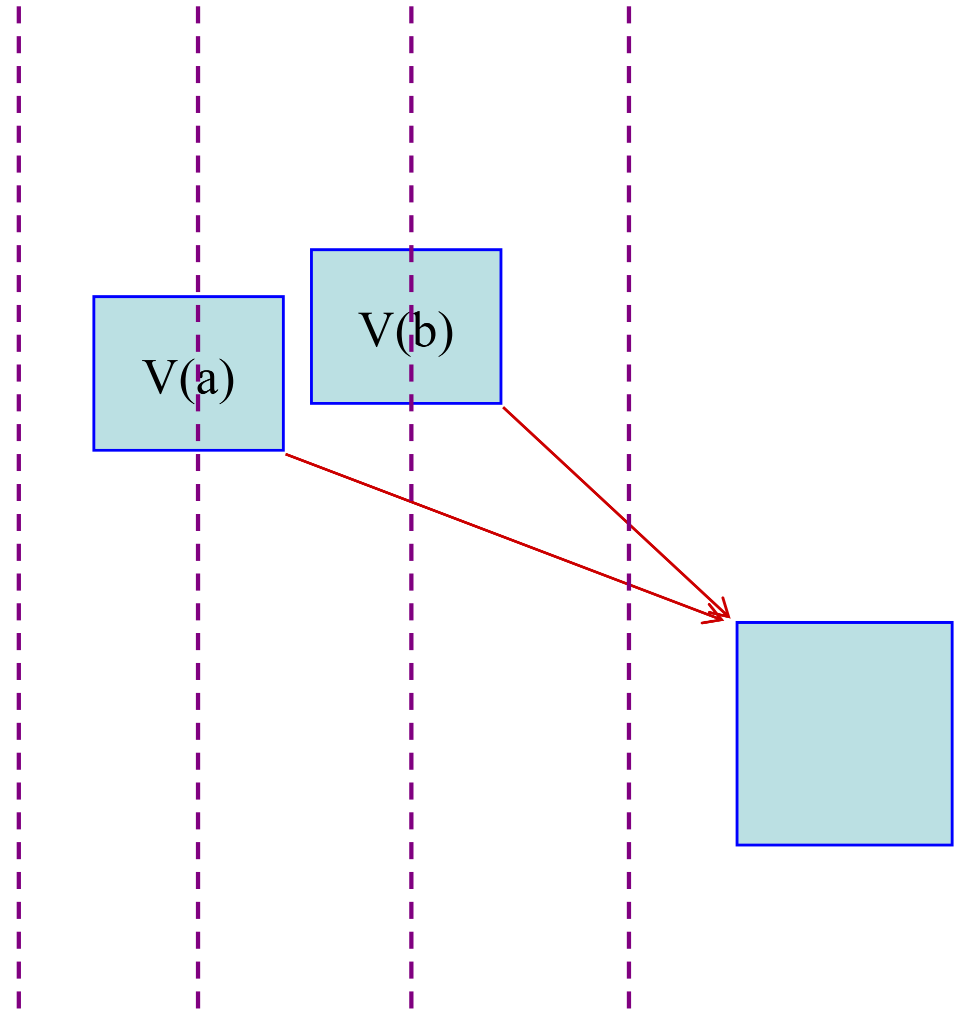


Slide from Serafim Batzoglou, Stanford University

Sparse DP for rectangle chaining

Main idea:

- Sweep through x-coordinates
- To the right of **b**, anything chainable to **a** is chainable to b
- Therefore, if **V(b) > V(a)**, rectangle a is "useless" for subsequent chaining
- In L, keep rectangles j sorted with increasing l_j-coordinates ⇒
  sorted with increasing V(j) score



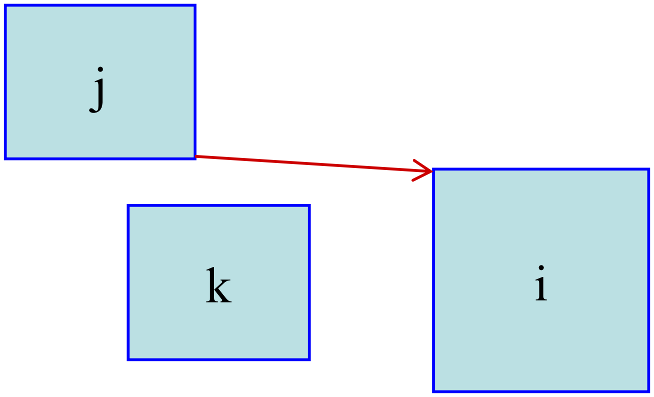
Slide from Serafim Batzoglou, Stanford University

Sparse DP for rectangle chaining

Go through rectangle x-coordinates, from lowest to highest:

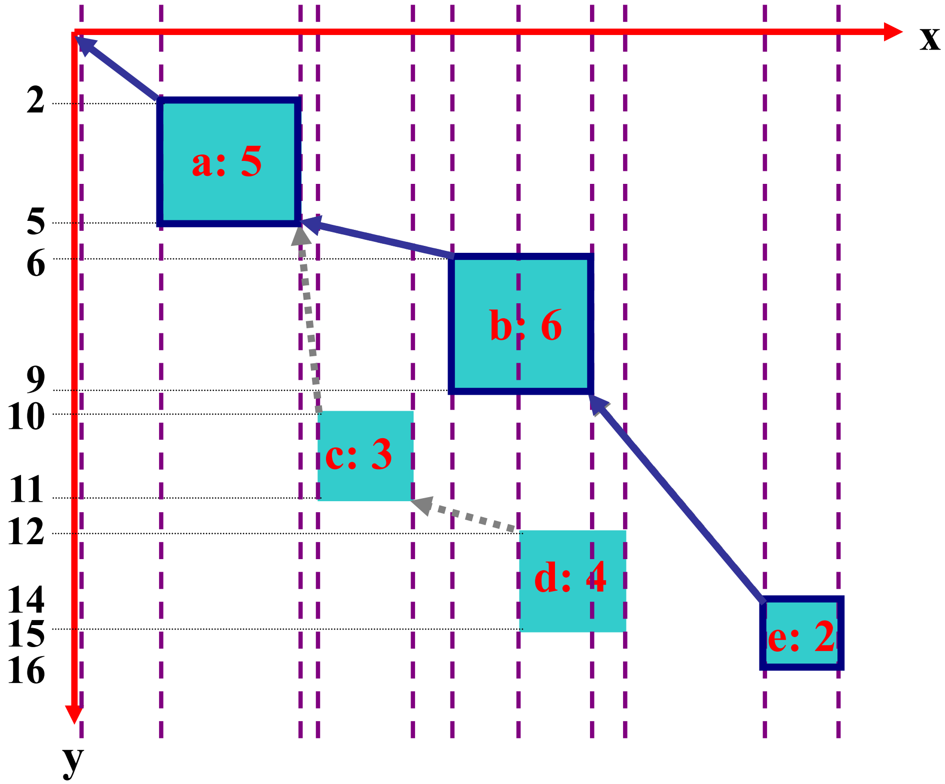
1. When on the leftmost end of rectangle i:
   a. j: rectangle in L, with largest $l_j < h_i$
   b. $V(i) = w(i) + V(j)$

2. When on the rightmost end of i:
   a. k: rectangle in L, with largest $l_k \leq l_i$
   b. If $V(i) > V(k)$:
      i. **INSERT** $(l_i, V(i), i)$ in L
      ii. **REMOVE** all $(l_j, V(j), j)$ with $V(j) \leq V(i)$ & $l_j \geq l_i$



Slide from Serafim Batzoglou, Stanford University

Example



V

a	b	c	d	e
5	11	8	12	13

L

l_i	5	9	15	16
V(i)	5	11	12	13
i	a	b	d	e

1. When on the leftmost end of rectangle i:
   a. j: rectangle in L, with largest $l_j < h_i$
   b. V(i) = w(i) + V(j)
2. When on the rightmost end of i:
   a. k: rectangle in L, with largest $l_k \leq l_i$
   b. If V(i) > V(k):
      i. **INSERT** $(l_i$, V(i), i) in L
      ii. **REMOVE** all $(l_j$, V(j), j) with $V(j) \leq V(i)$ & $l_j \geq l_i$

Slide from Serafim Batzoglou, Stanford University

Time Analysis

1. Sorting the x-coords takes O(N log N)

2. Going through x-coords: N steps

3. Each of N steps requires O(log N) time:

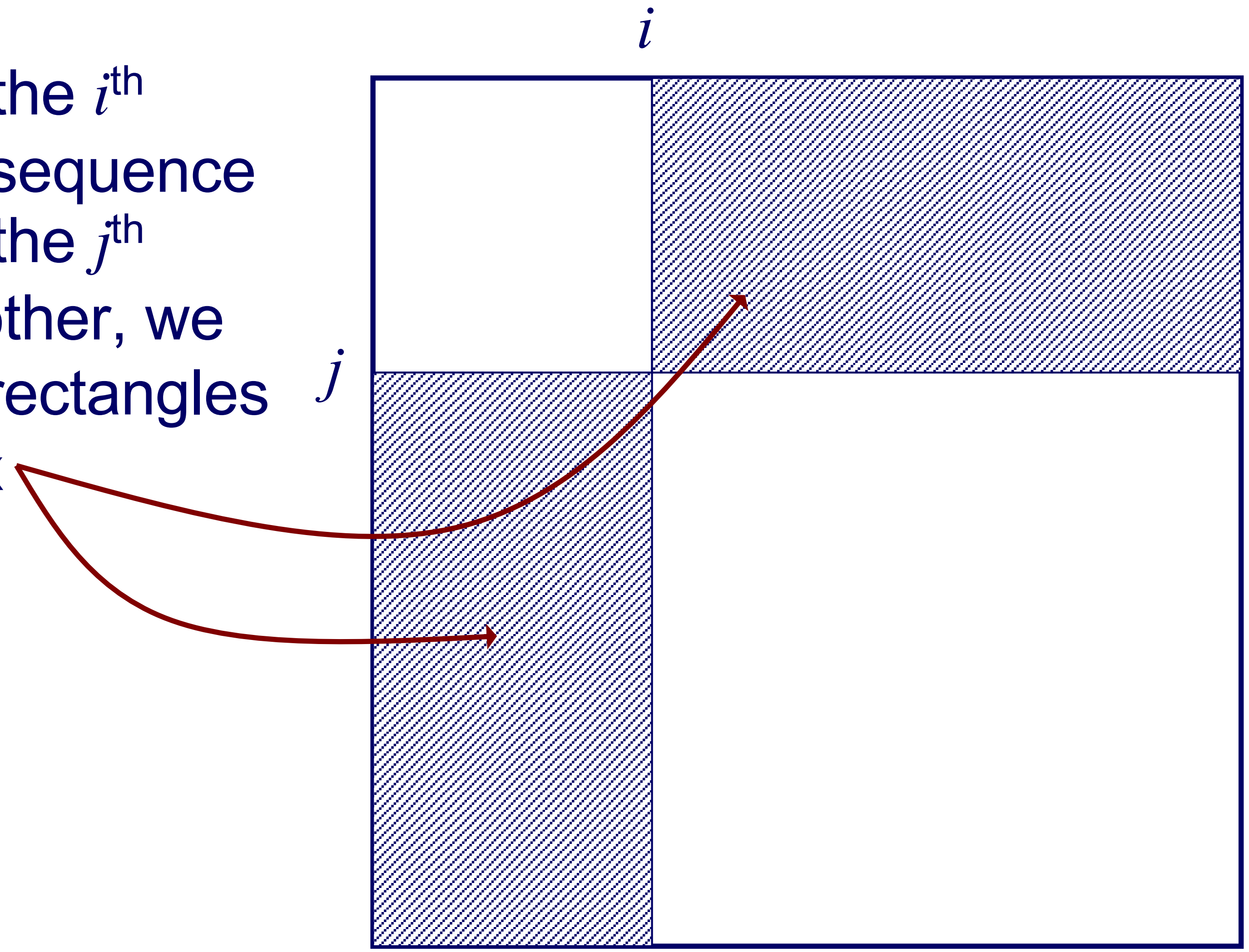
   - Searching L takes log N
   - Inserting to L takes log N
   - All deletions are consecutive, so log N per deletion
   - Each element is deleted at most once: N log N for all deletions

     - Recall that INSERT, DELETE, SUCCESSOR, take O(log N) time in a balanced binary search tree

Slide from Serafim Batzoglou, Stanford University

Constrained Dynamic Programming

- if we know that the i^{th} element in one sequence must align with the j^{th} element in the other, we can ignore two rectangles in the DP matrix



Step 3: Computing the Global Alignment in LAGAN

- given an anchor that starts at (i, j) and ends at (i', j'), LAGAN limits the DP to the unshaded regions
- thus anchors are somewhat flexible

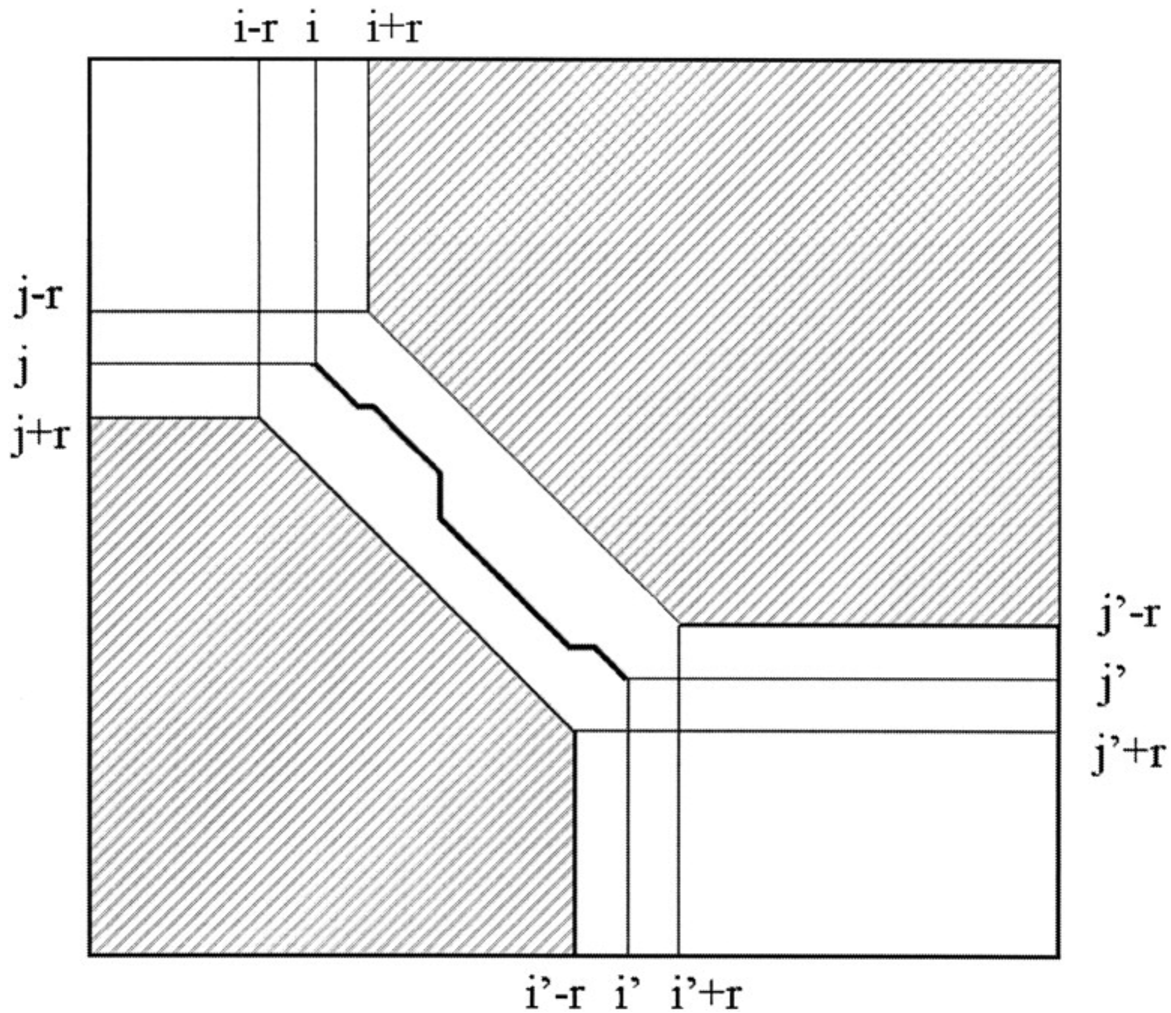
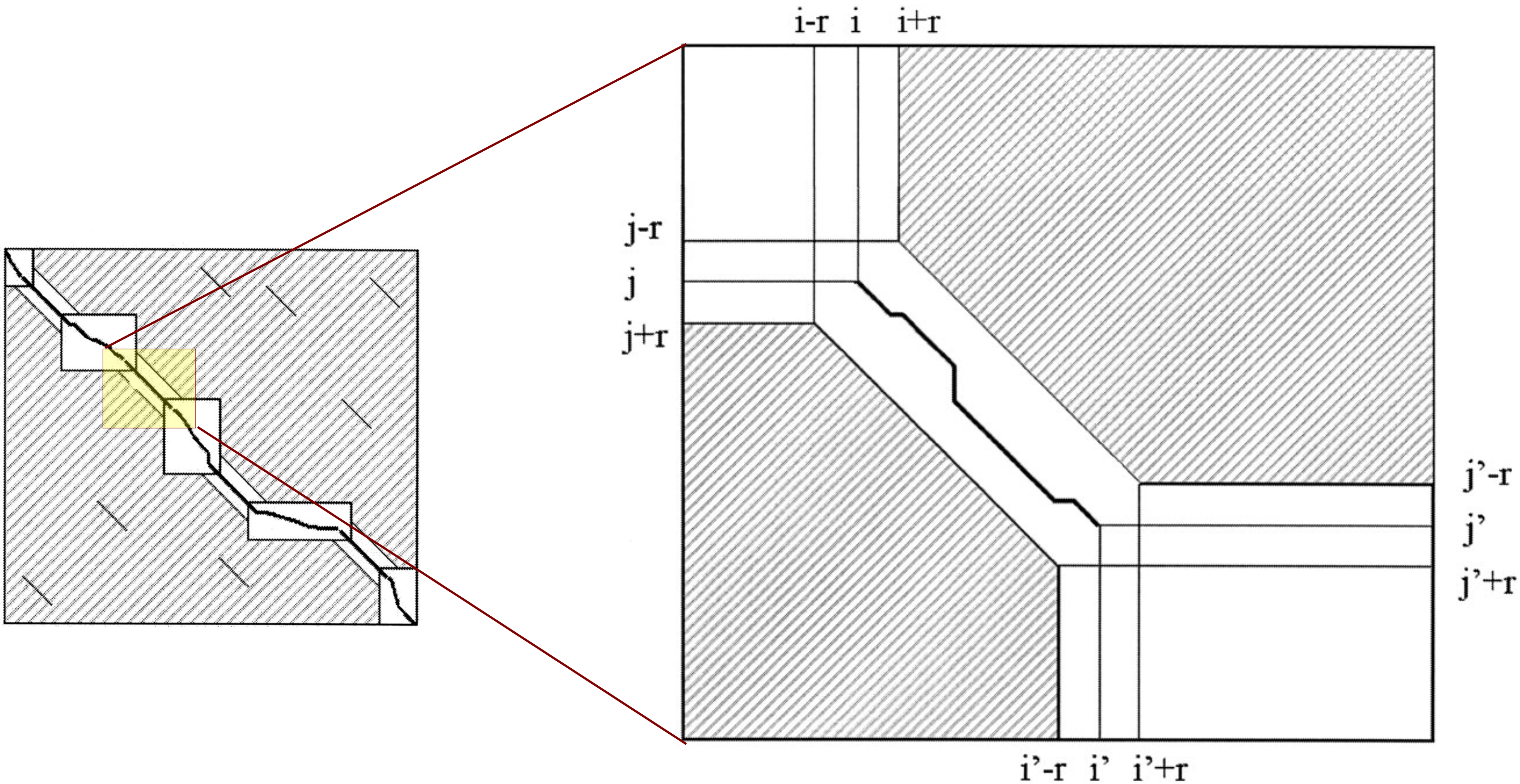


Figure from: Brudno et al. *Genome Research*, 2003

Step 3: Computing the Global Alignment in LAGAN

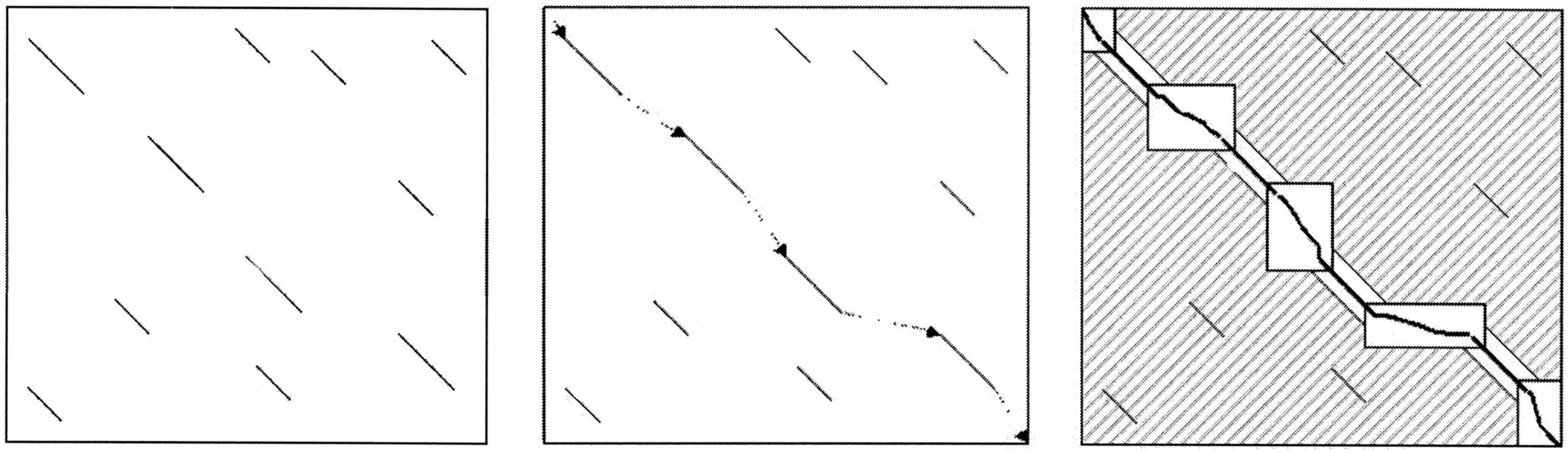


Figures from: Brudno et al. *Genome Research*, 2003

Comparing MUMmer and LAGAN

	Baboon	Chimpanzee	Mouse	Rat	Cow	Pig	Cat	Dog	Chicken	Zebrafish	Fugu	Overall
Exons	232	176	230	230	224	174	176	182	68	48	150	1914
MUMmer (% human exons covered by ≥ 90% alignment)	100	100	8	9	40	44	47	37	0	0	0	41
LAGAN (% human exons covered by ≥ 90% alignment)	100	100	100	100	99	100	100	99	99	88	77	98

Comparing MUMmer and LAGAN



Brudno et al. *Genome Research*, 2003

1. Pattern matching to find seeds for global alignment
2. Find a good chain of anchors
3. Fill in with standard but constrained alignment

MUMmer

1. Suffix trees to obtain MUMs
2. Longest Increasing Subsequence
3. Smith-Waterman, recursive MUMmer

LAGAN

1. k-mer trie to obtain seeds
2. Sparse dynamic programming
3. Dynamic programming