Alignment of Long Sequences

BMI/CS 776
www.biostat.wisc.edu/bmi776/
Spring 2009
Mark Craven
craven@biostat.wisc.edu

Pairwise Whole Genome Alignment: Task Definition

- Given
  - a pair of genomes (or other large-scale sequences)
  - a method for scoring the similarity of a pair of characters
- Do
  - construct global alignment: identify matches between genomes as well as various non-match features

Example: *E. Coli* O157:H7 vs. E. coli K-12

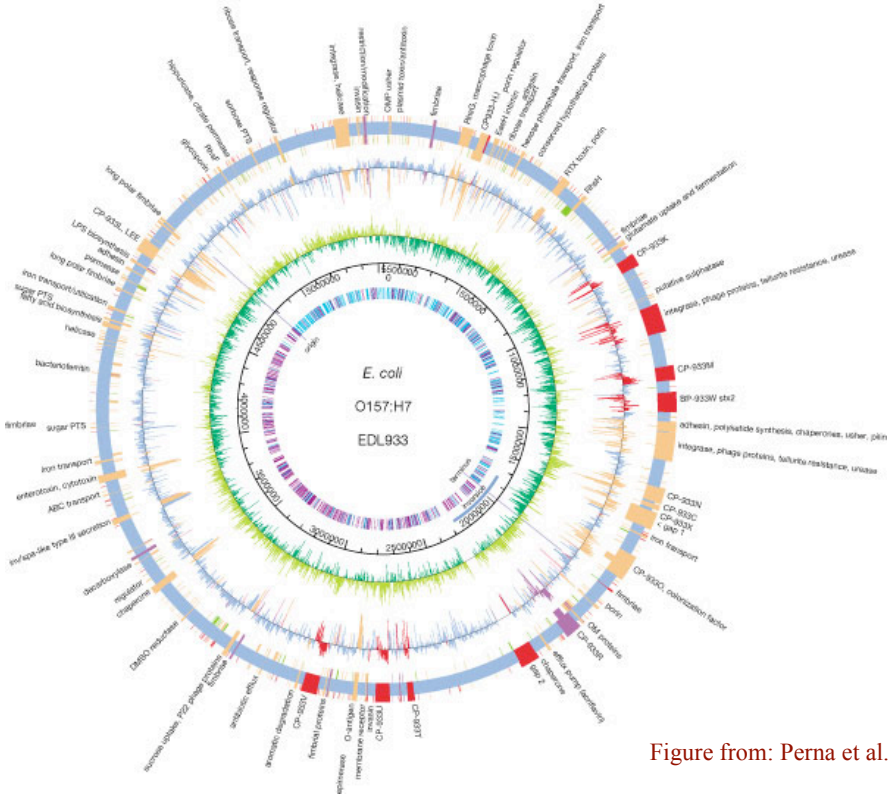
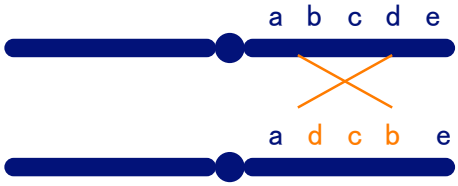


Figure from: Perna et al. *Nature*, 2001

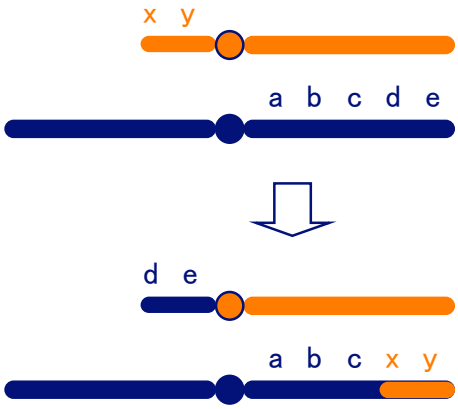
Why Not Use Standard Dynamic Programming Methods?

- sequences too big to make $O(n^2)$ methods practical
- sequences may involve genome *rearrangements*
  - standard alignment methods account for
    - point mutations
    - short insertions and deletions
  - whole genome methods must also consider
    - inversions
    - translocations
    - large insertions and deletions (e.g. from horizontal transfer)

Genome Rearrangements



inversion

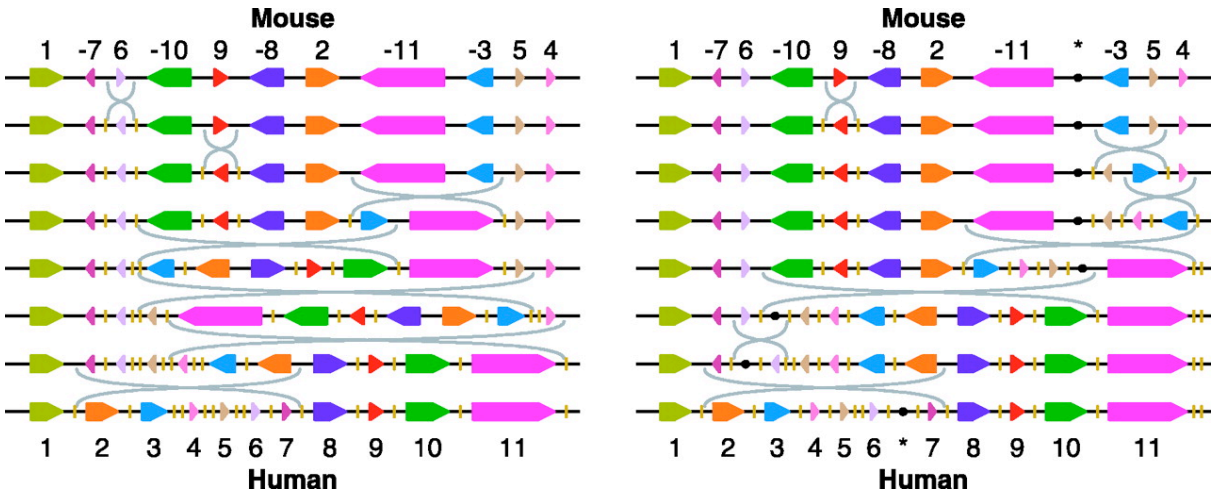


translocation

- can occur within a chromosome or across chromosomes
- can have combinations of these events

Genome Rearrangement Example: Mouse vs. Human X Chromsome

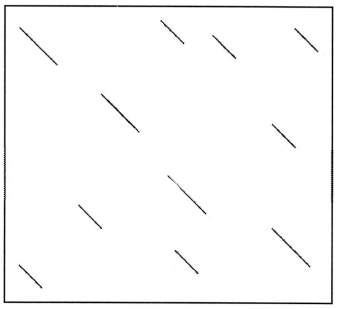
Figure from: Pevzner and Tesler. *PNAS*, 2003



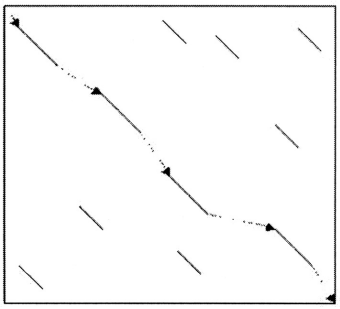
- each colored block represents a syntenic region of the two chromosomes
- the two panels show the two most parsimonious sets of rearrangements to map one chromosome to the other

Large Scale Alignment Illustrated

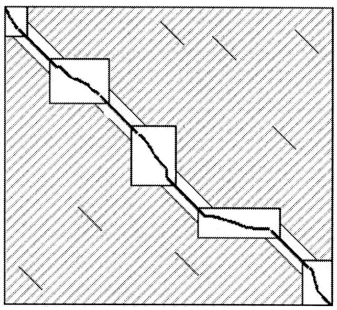
Figure from: Brudno et al. *Genome Research*, 2003



1. perform pattern matching to find seeds for global alignment



2. find a good chain of anchors



3. fill in remainder with standard but constrained alignment method

Method Comparison

Method	Pattern matching	Chaining
MUMmer	Suffix tree - MUMs	LIS variant
AVID	Suffix tree - exact & wobble matches	Smith-Waterman variant
LAGAN	k-mer trie, inexact matches	Sparse DP

The MUMmer System

- Delcher et al., *Nucleic Acids Research*, 1999

- Given: genomes *A* and *B*

  find all maximal, unique, matching subsequences (MUMs)

  extract the longest possible set of matches that occur in the same order in both genomes

  close the gaps

  output the alignment

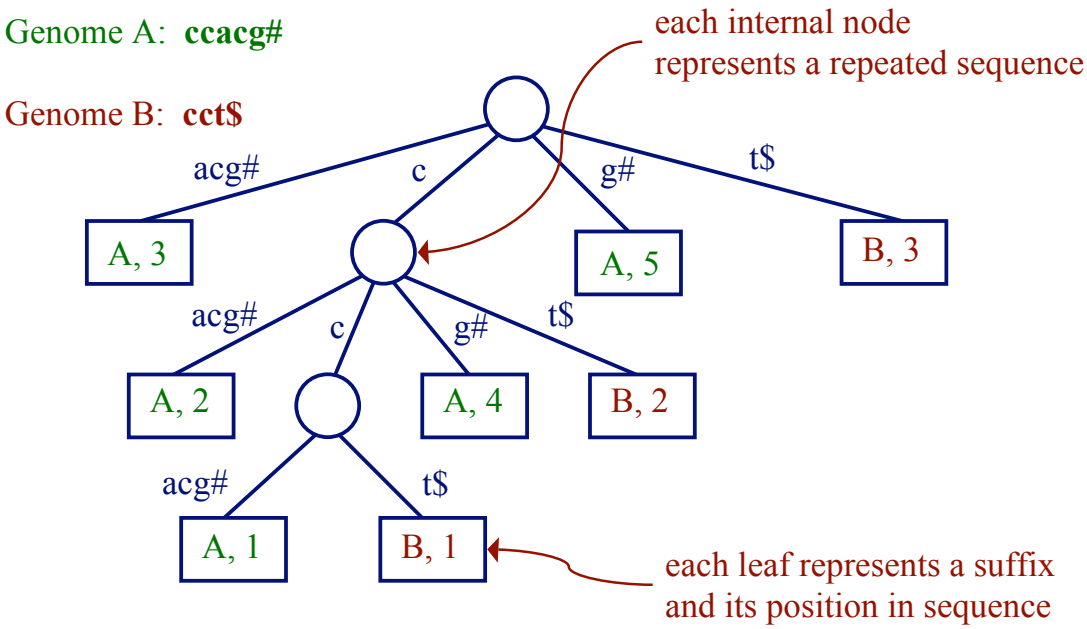
Step 1: MUM Decomposition

- *maximal unique match* (MUM):
  - occurs exactly once in both genomes *A* and *B*
  - not contained in any longer MUM

- key insight: a significantly long MUM is certain to be part of the global alignment

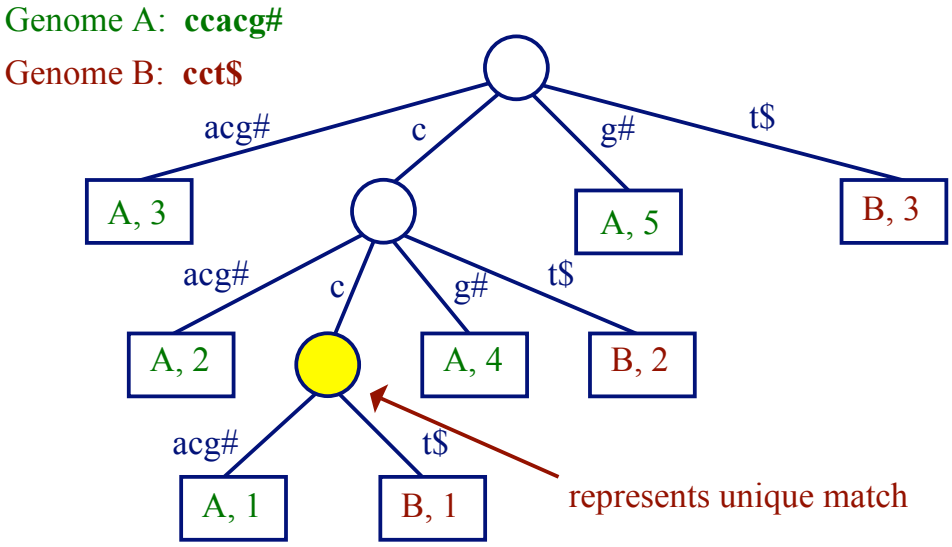
MUMs and *Generalized* Suffix Trees

- add suffixes for both genomes *A* and *B* to tree
- label each leaf node with genome it represents



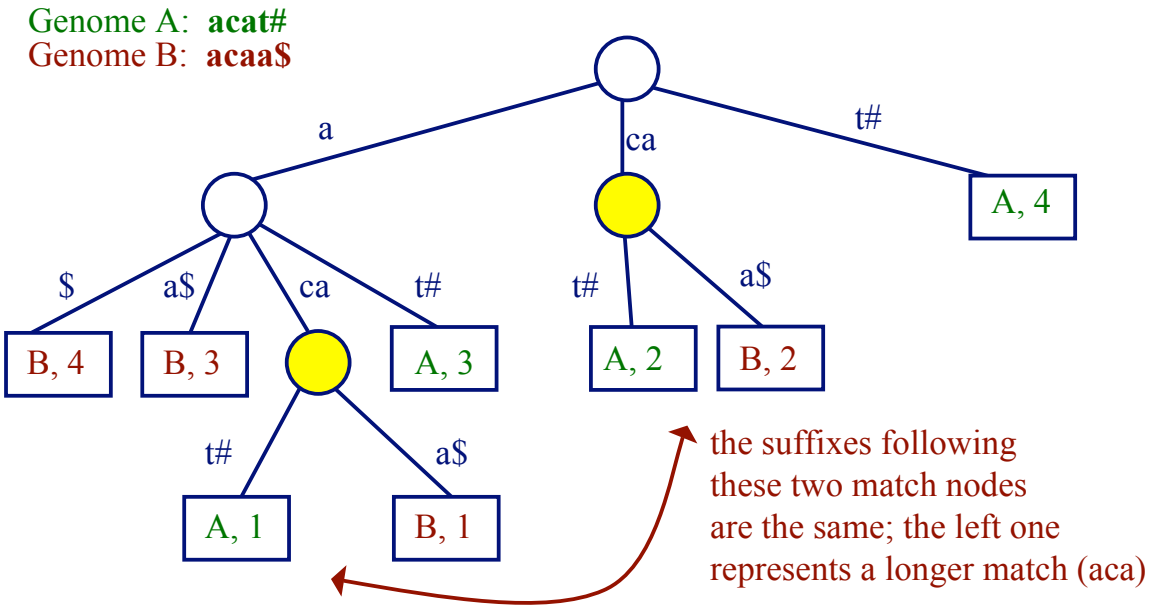
MUMs and Suffix Trees

- unique match: internal node with 2 children, leaf nodes from different genomes
- but these matches are not necessarily maximal



MUMs and Suffix Trees

- to identify maximal matches, can compare suffixes following unique match nodes



Using Suffix Trees to Find MUMs

- can build in linear time (in lengths of genomes)
- can identify all MUMs in linear time (one scan of tree)
- space complexity is linear (exactly one leaf and at most one internal node for each base)
- main parameter of system: length of shortest MUM that should be identified (20 – 50 bases)

MUM Complexity

- O(n) time to construct suffix tree for both sequences (of lengths < n)
- O(n) time to find MUMs - one scan of the tree (which is O(n) in size)
- O(n) possible MUMs in contrast to $O(n^2)$ possible exact matches

Step 2: Find Longest Subsequence

- sort MUMs according to position in genome A
- solve variation of *Longest Increasing Subsequence* (LIS) problem to find sequences in ascending order in both genomes

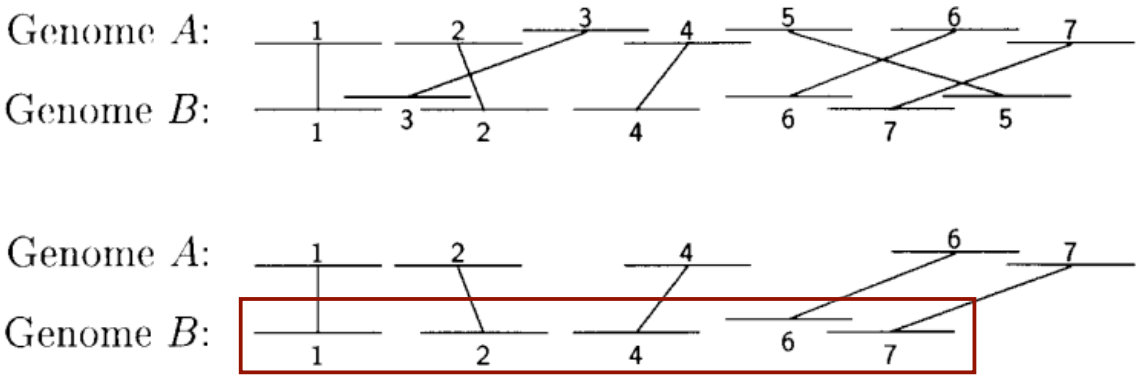


Figure from: Delcher et al., *Nucleic Acids Research* 27, 1999

Finding Longest Subsequence

- unlike ordinary LIS problems, MUMmer takes into account
  - lengths of sequences represented by MUMs
  - overlaps
- requires $O(k \log k)$ time where k is number of MUMs

Types of Gaps in a MUMmer Alignment

1. SNP: exactly one base (indicated by ^) differs between the two sequences. It is surrounded by exact-match sequence.

```
Genome A:  cgtcatgggcgttcgtcgttg
Genome B:  cgtcatgggcattcgtcgttg
                     ^
```

2. Insertion: a sequence that occurs in one genome but not the other.

```
Genome A:  cggggtaaccgc.................cctggtcggg
Genome B:  cggggtaaccgcgttgctcggggtaaccgccctggtcggg
                     ^^^^^^^^^^^^^^^^^^
```

3. Highly polymorphic region: many mutations in a short region.

```
Genome A:  ccgcctcgcctgg.gctggcgcccgctc
Genome B:  ccgcctcgccagttgaccgcgcccgctc
                   ^ ^^ ^^^
```

4. Repeat sequence: the repeat is shown in uppercase. Note that the first copy of the repeat in Genome B is imperfect, containing one mismatch to the other three identical copies.

```
Genome A:  cTGGGTGGGACAACGTaaaaaaaaaTGGGTGGGACAACGTc
Genome B:  aTGGGTGGGGCgACGTgggggggggTGGGTGGGACAACGTa
           ^           ^                          ^
```

Figure from: Delcher et al., *Nucleic Acids Research* 27, 1999

Step 3: Close the Gaps

- SNPs:
  - between MUMs: trivial to detect
  - otherwise: handle like repeats
- inserts
  - transpositions (subsequences that were deleted from one location and inserted elsewhere): look for out-of-sequence MUMs
  - simple insertions: trivial to detect

Step 3: Close the Gaps

- polymorphic regions
  - short ones: align them with dynamic programming method
  - long ones: call MUMmer recursively w/ reduced min MUM length
- repeats
  - detected by overlapping MUMs

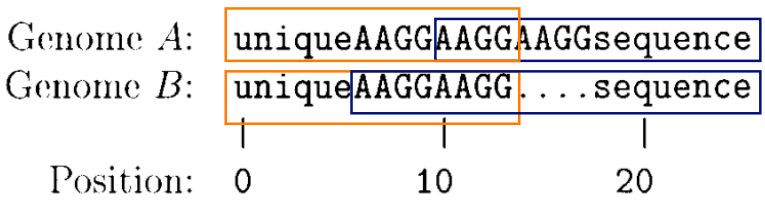


Figure from: Delcher et al. Nucleic Acids Research 27, 1999

The LAGAN Method

Brudno et al., *Genome Research*, 2003

Given: genomes *A* and *B*

anchors = find_anchors(A, B)

`step 3`: finish global alignment with DP constrained by *anchors*

find_anchors(A, B)

`step 1`: find local alignments by matching, chaining *k*-mer seeds

`step 2`: *anchors* = highest-weight sequence of local alignments

for each pair of adjacent anchors a_1, a_2 in *anchors*

if a_1, a_2 are more than *d* bases apart

A', *B'* = sequences between a_1, a_2

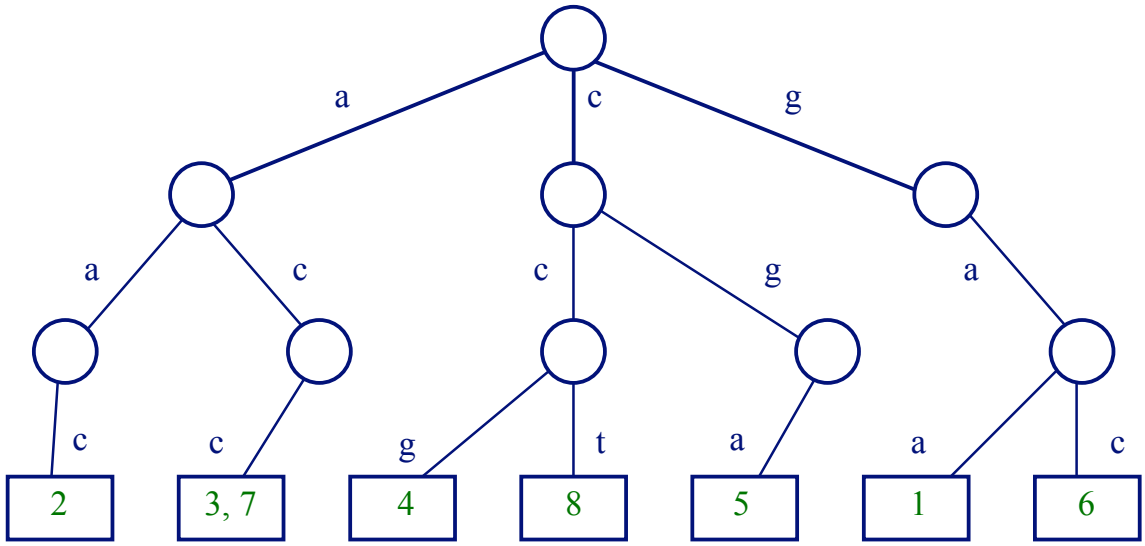
sub-anchors = find_anchors(*A'*, *B'*)

insert *sub-anchors* between a_1, a_2 in *anchors*

return *anchors*

Step 1a: Using *Tries* to Find Seeds

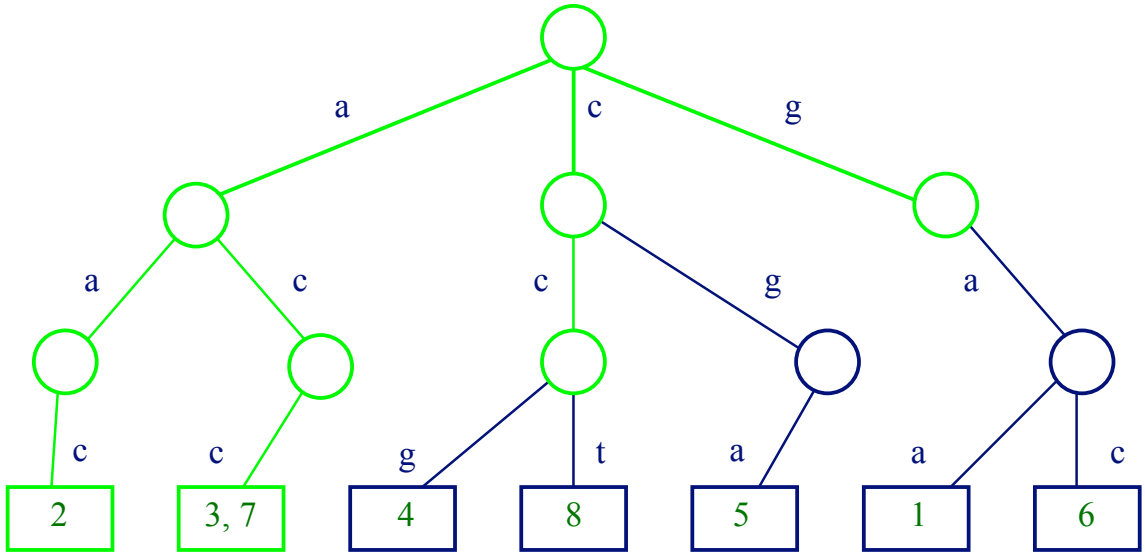
- a *trie* to represent all 3-mers of the sequence **gaaccgacct**



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

Allowing Degenerate Matches

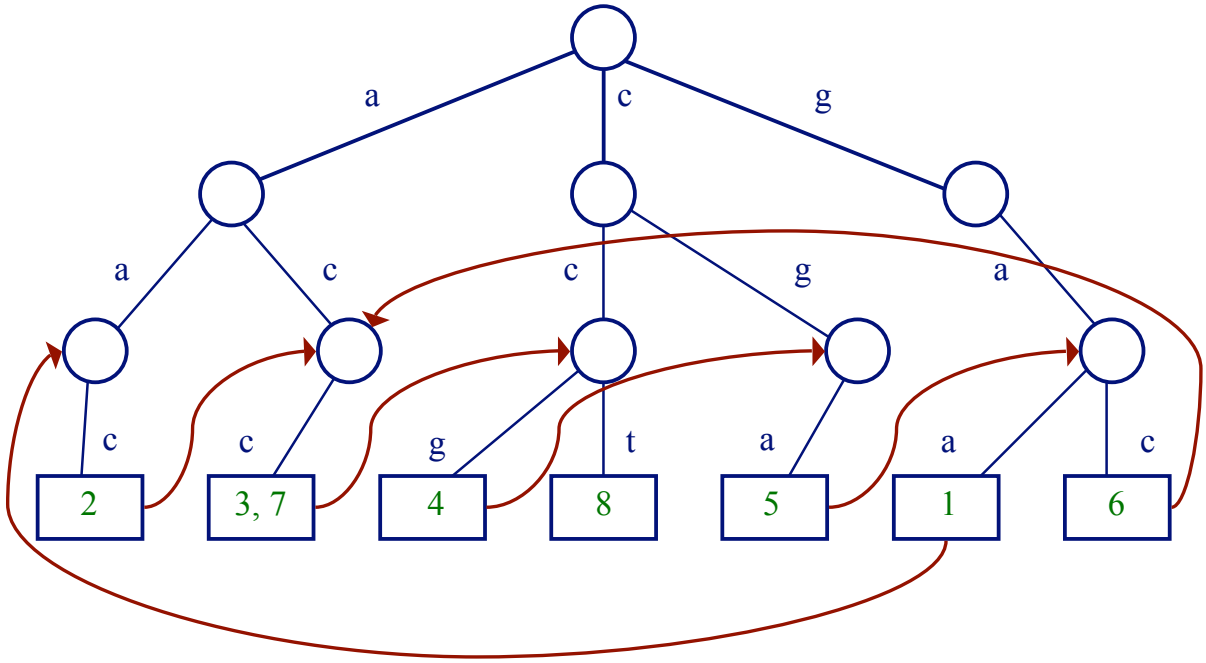
- suppose we're allowing 1 base to mismatch in looking for matches to the 3-mer **acc**; need to explore green nodes



- by default, LAGAN uses 10-mers and allows 1 mismatch

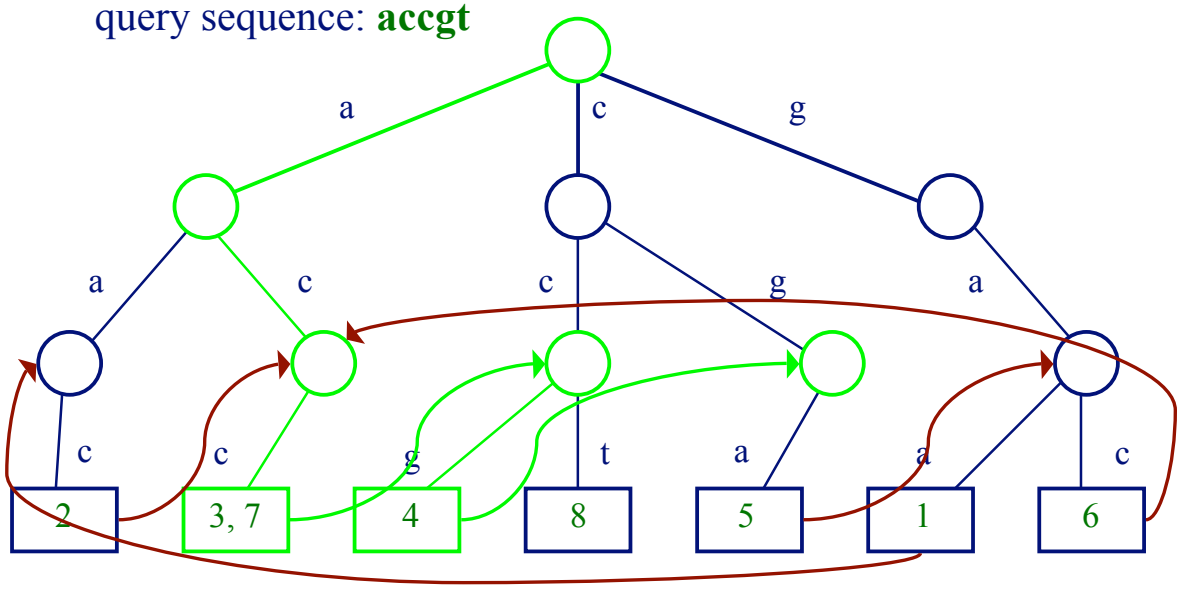
LAGAN Uses Threaded Tries

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



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

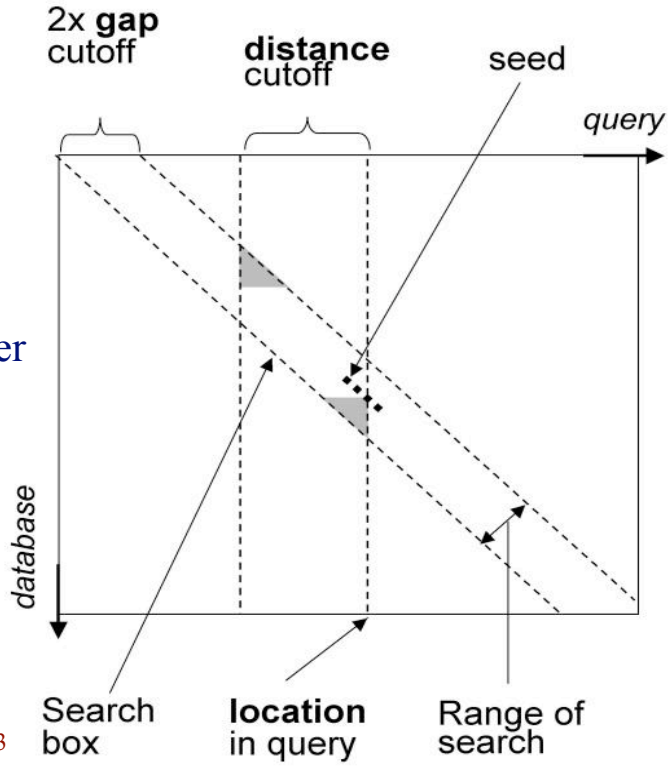


Figure from: Brudno et al. *BMC Bioinformatics*, 2003

Step 2: Find Longest Subsequence

- like MUMmer, solve variation of Longest Increasing Subsequence (LIS) problem to find chained seeds in ascending order in both genomes

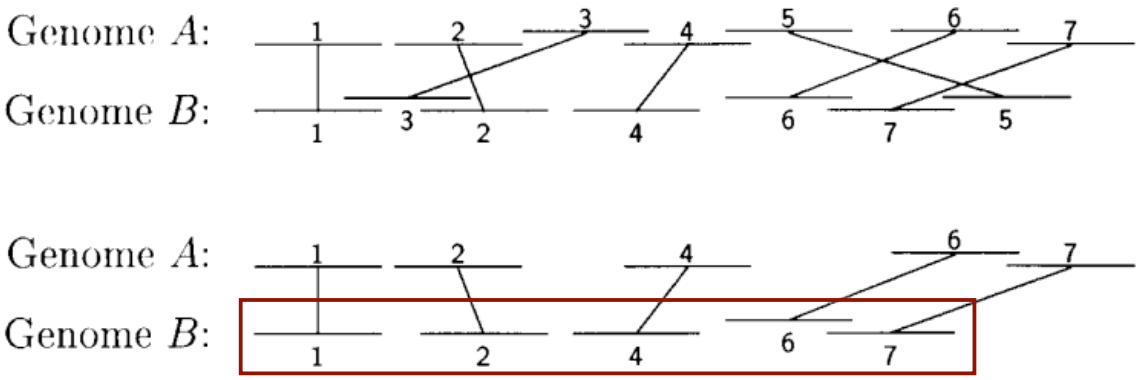
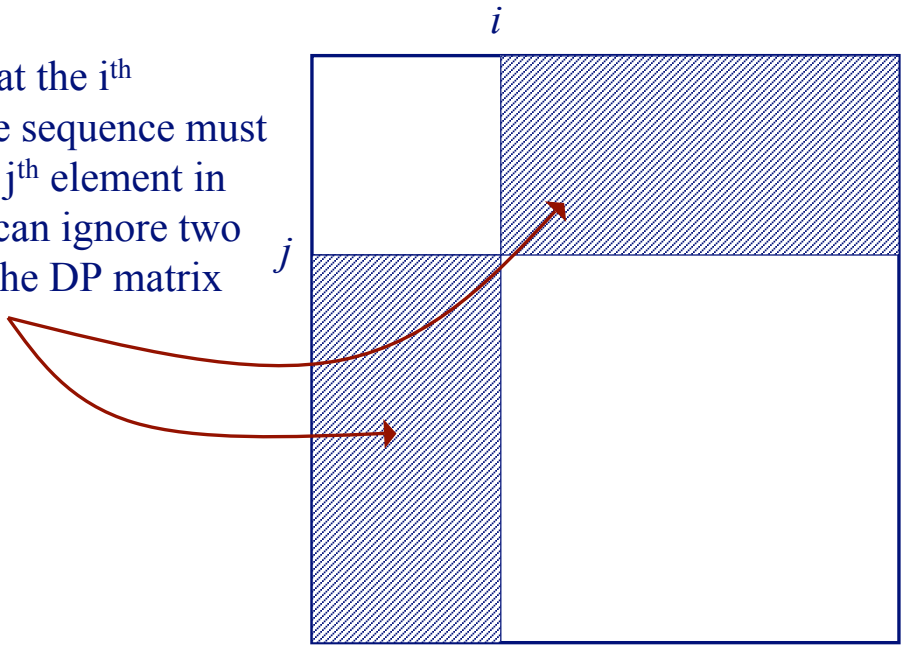


Figure from: Delcher et al. Nucleic Acids Research 27, 1999

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

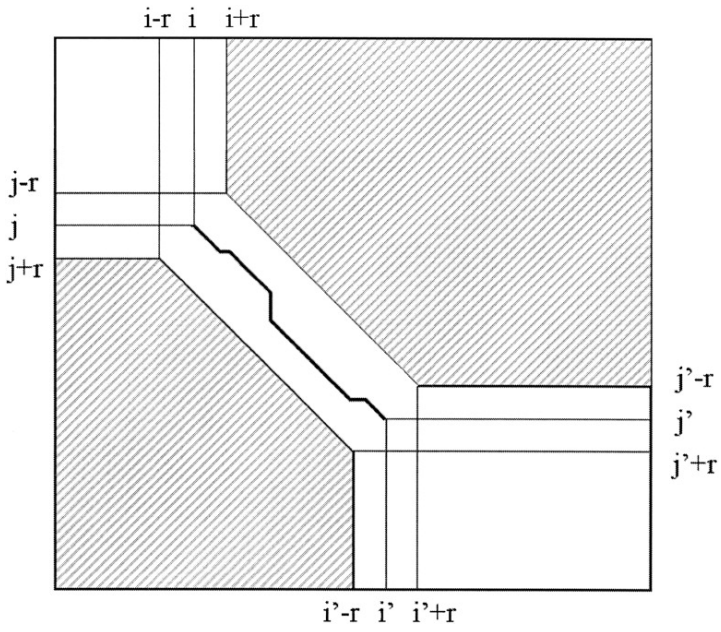
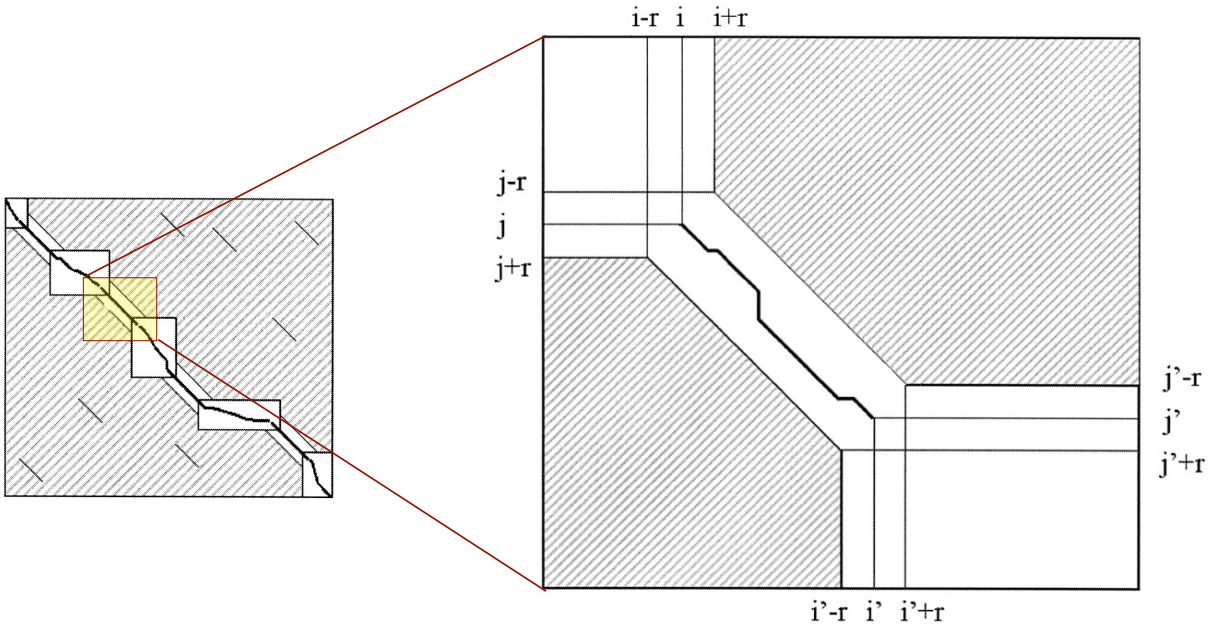


Figure from: Brudno et al. *Genome Research*, 2003

Step 3: Computing the Global Alignment in LAGAN



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

The AVID Method

- RepeatMask sequences
- find anchors (suffix tree, exact & wobble)
- find good chain of anchors (Smith-Waterman variant)
- for each inter-anchor region, is the region small enough to do base-pair alignment?

  yes - Run Needleman-Wunsch on region

  no - Recurse starting at anchor chaining step

Anchors in AVID

- all maximal exact matches > some minimum length
  - suffix tree construction + traversal
- divide matches into “clean” or “repeat” depending on whether intervals overlap a repetitive element (annotated by RepeatMasker)
  - repeat matches used only after all clean matches are considered
- also locate “wobble” matches
  - inexact matches, possibly mismatching at every third base

"Wobble" Bases in Codons

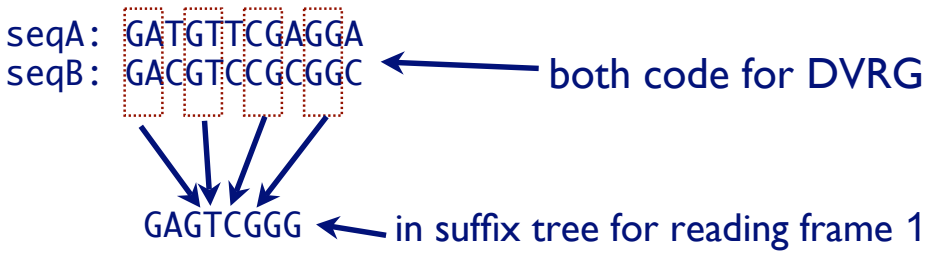
- substitutions in 3rd codon position often do not change amino acid encoded

Second base

First base	U	C	A	G	Third base
U	UUU, UUC — Phenylalanine F UUA, UUG — Leucine L	UCU, UCC, UCA, UCG — Serine S	UAU, UAC — Tyrosine Y UAA Stop codon, UAG Stop codon	UGU, UGC — Cysteine C UGA Stop codon, UGG Tryptophan W	U C A G
C	CUU, CUC, CUA, CUG — Leucine L	CCU, CCC, CCA, CCG — Proline P	CAU, CAC — Histidine H CAA, CAG — Glutamine Q	CGU, CGC, CGA, CGG — Arginine R	U C A G
A	AUU, AUC, AUA — Isoleucine I AUG Methionine start codon M	ACU, ACC, ACA, ACG — Threonine T	AAU, AAC — Asparagine N AAA, AAG — Lysine K	AGU, AGC — Serine S AGA, AGG — Arginine R	U C A G
G	GUU, GUC, GUA, GUG — Valine V	GCU, GCC, GCA, GCG — Alanine A	GAU, GAC — Aspartic acid D GAA, GAG — Glutamic acid E	GGU, GGC, GGA, GGG — Glycine G	U C A G

Wobble Matches

- trick for better alignment of protein-coding DNA
- look for exact matches ignoring every 3rd base
- build suffix tree for all 3 reading frames



Chaining Anchors in AVID via SW

- assign a unique character to each set of anchor sequences
- replace input DNA sequences by sequence of anchor characters
- perform Smith-Waterman on anchor character sequences
  - gap penalty = 0, mismatch = $-\infty$
  - match score = score of local alignment around anchor