Alignment of Long Sequences

BMI/CS 776

www.biostat.wisc.edu/bmi776/

Spring 2018

Anthony Gitter

gitter@biostat.wisc.edu

Goals for Lecture

Key concepts

- how large-scale alignment differs from the simple case
- the canonical three step approach of large-scale aligners
- using suffix trees to find maximal unique matching subsequences (MUMs)

If time permits

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

Pairwise Large-Scale Alignment: Task Definition

Given

- a pair of large-scale sequences (e.g. chromosomes)
- a method for scoring the alignment (e.g. substitution matrices, insertion/deletion parameters)

Do

- construct global alignment: identify all matching positions between the two sequences

Large Scale Alignment Example

Mouse Chr6 vs. Human Chr12

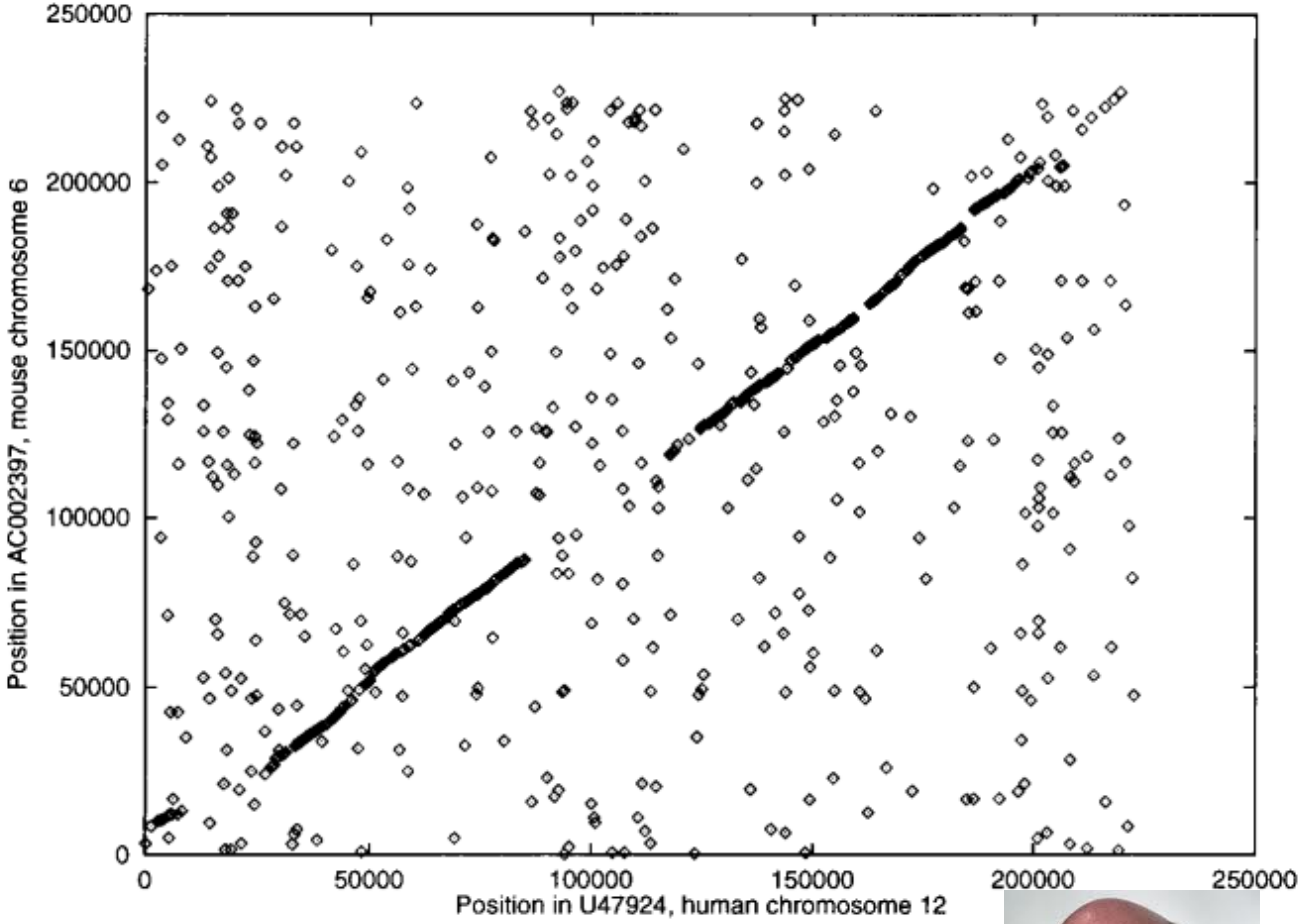


Figure from: Delcher et al., *Nucleic Acids Research* 27, 1999

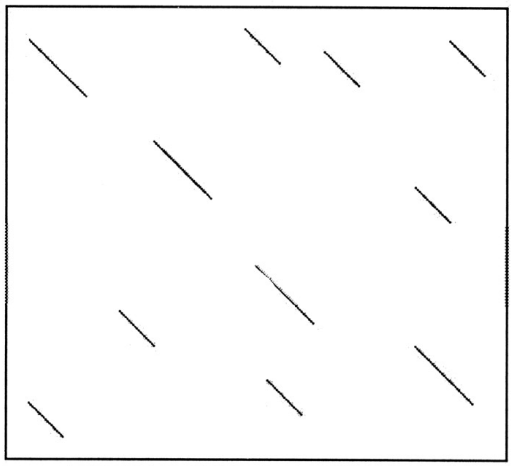
Why the Problem is Challenging

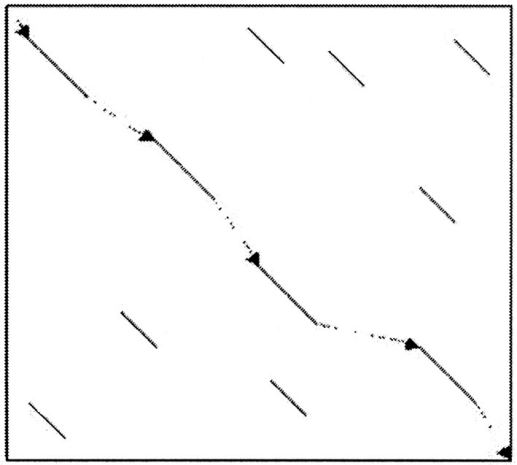
- Sequences too big to make $O(n^2)$ dynamic-programming methods practical

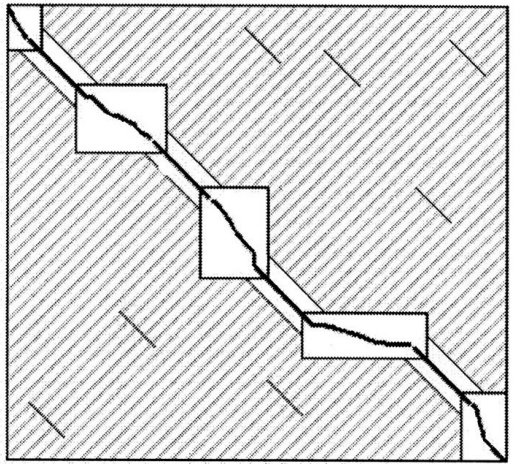
- Long sequences are less likely to be colinear because of *rearrangements*
  - initially we'll assume colinearity
  - we'll consider rearrangements in next lecture (or never)

General Strategy

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

The MUMmer System

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

Given: genomes A and B

1. find all maximal unique matching subsequences (MUMs)
2. extract the longest possible set of matches that occur in the same order in both genomes
3. close the gaps

Step 1: Finding Seeds in MUMmer

- *Maximal unique match*:
  - 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

Suffix Trees

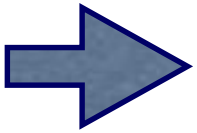
- Substring problem:
  - given text S of length m
  - preprocess S in $O(m)$ time
  - such that, given query string Q of length n, find occurrence (if any) of Q in S in $O(n)$ time

- Suffix trees solve this problem and others

Suffix Tree Definition

- A suffix tree T for a string S of length m is a tree with the following properties:
  - *rooted* and *directed*
  - m leaves, labeled 1 to m
  - each edge labeled by a substring of S
  - concatenation of edge labels on path from root to leaf i is suffix i of S (we will denote this by $S_{i...m}$)
  - each internal non-root node has at least two children
  - edges out of a node must begin with different characters

key property



Suffixes

S = “banana$”

suffixes of S

$ (special character)

a$

na$

ana$

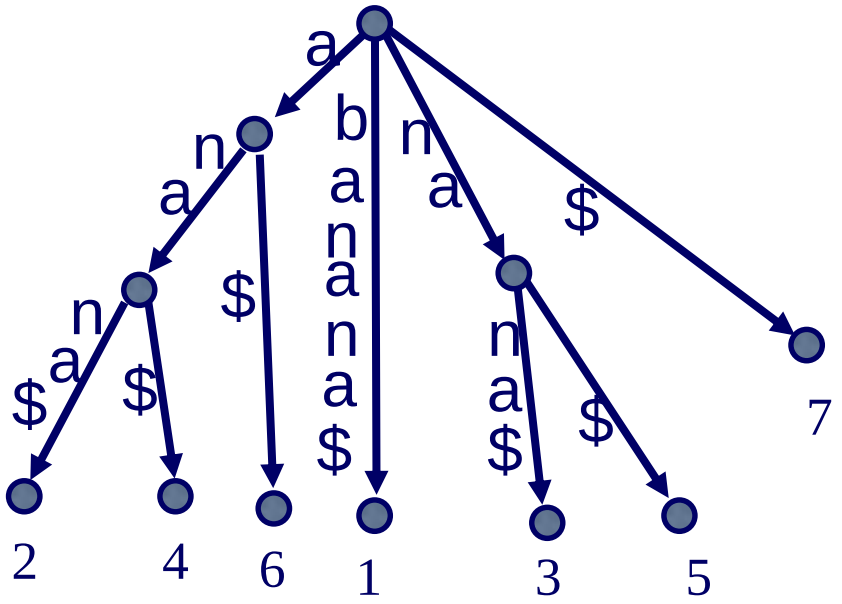
nana$

anana$

banana$

Suffix Tree Example

- S = “banana$”
- Add ‘$’ to end so that suffix tree exists (no suffix is a prefix of another suffix)



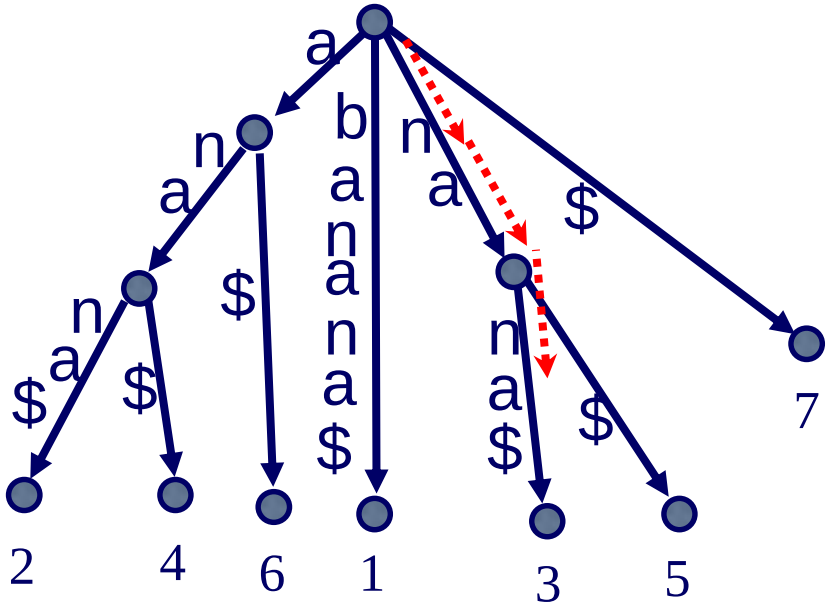
Solving the Substring Problem

- Assume we have suffix tree T and query string Q
- FindMatch(Q, T):
  - follow (unique) path down from root of T according to characters in Q
  - if all of Q is found to be a prefix of such a path

    return label of some leaf below this path
  - else, return no match found

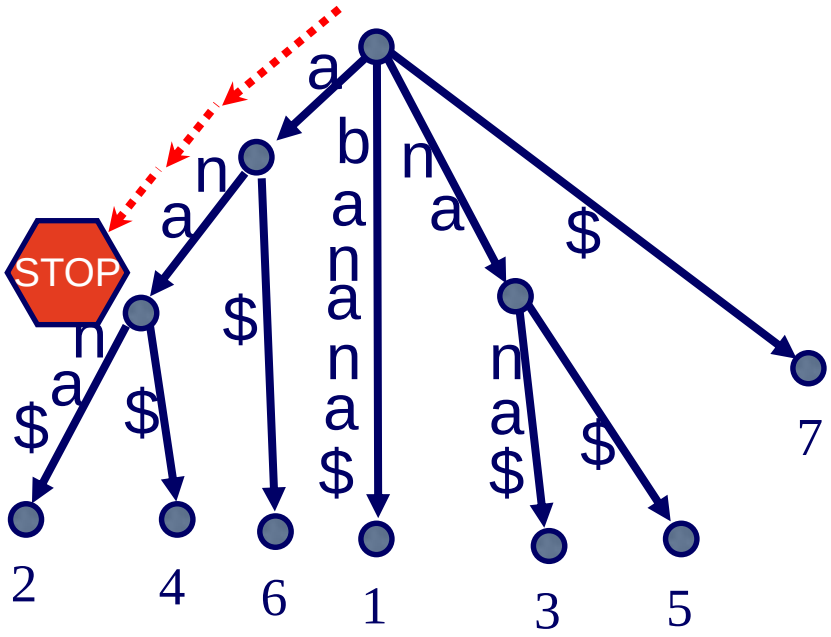
Solving the Substring Problem

Q = nan



return 3

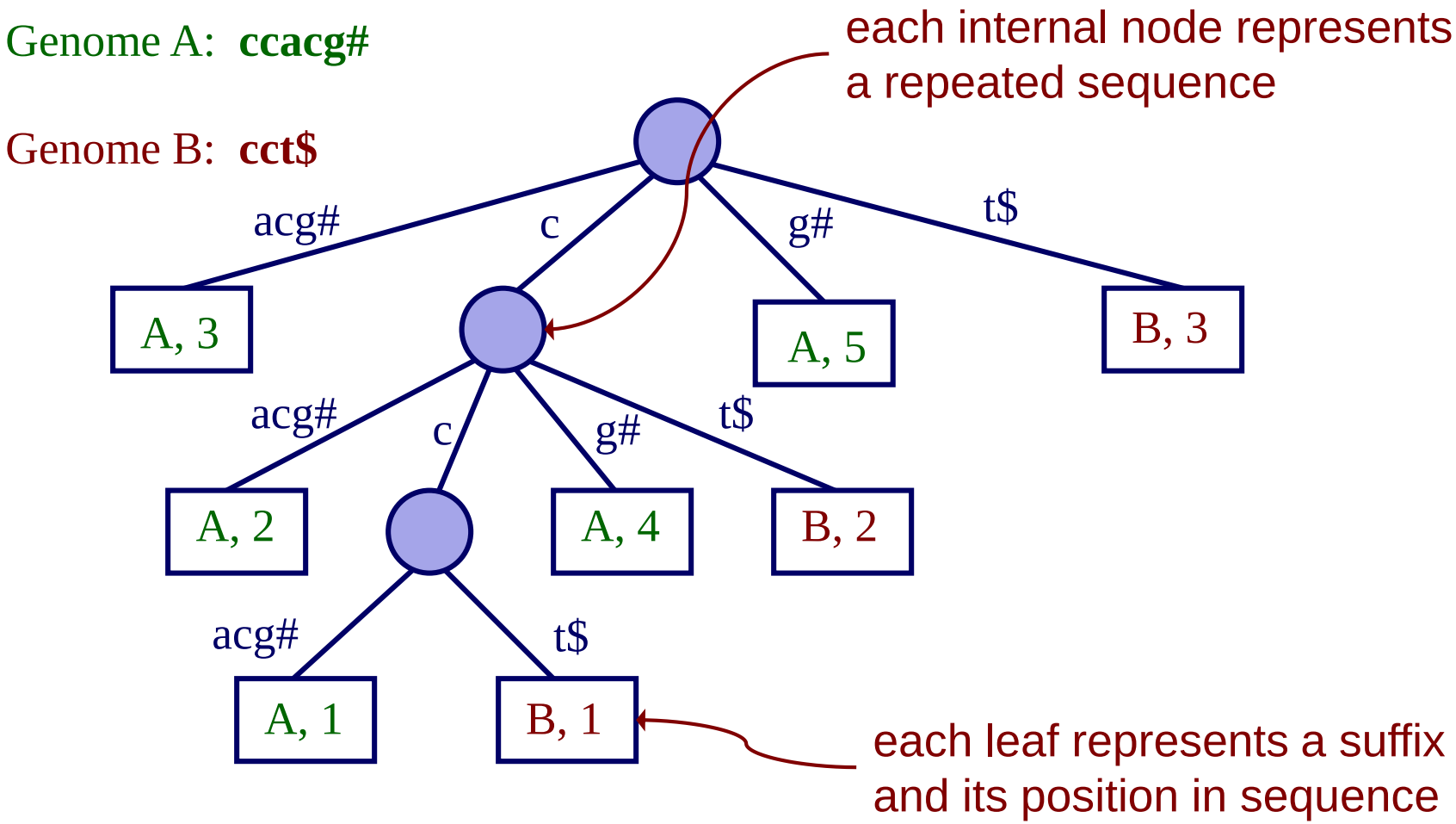
Q = anab



return no match found

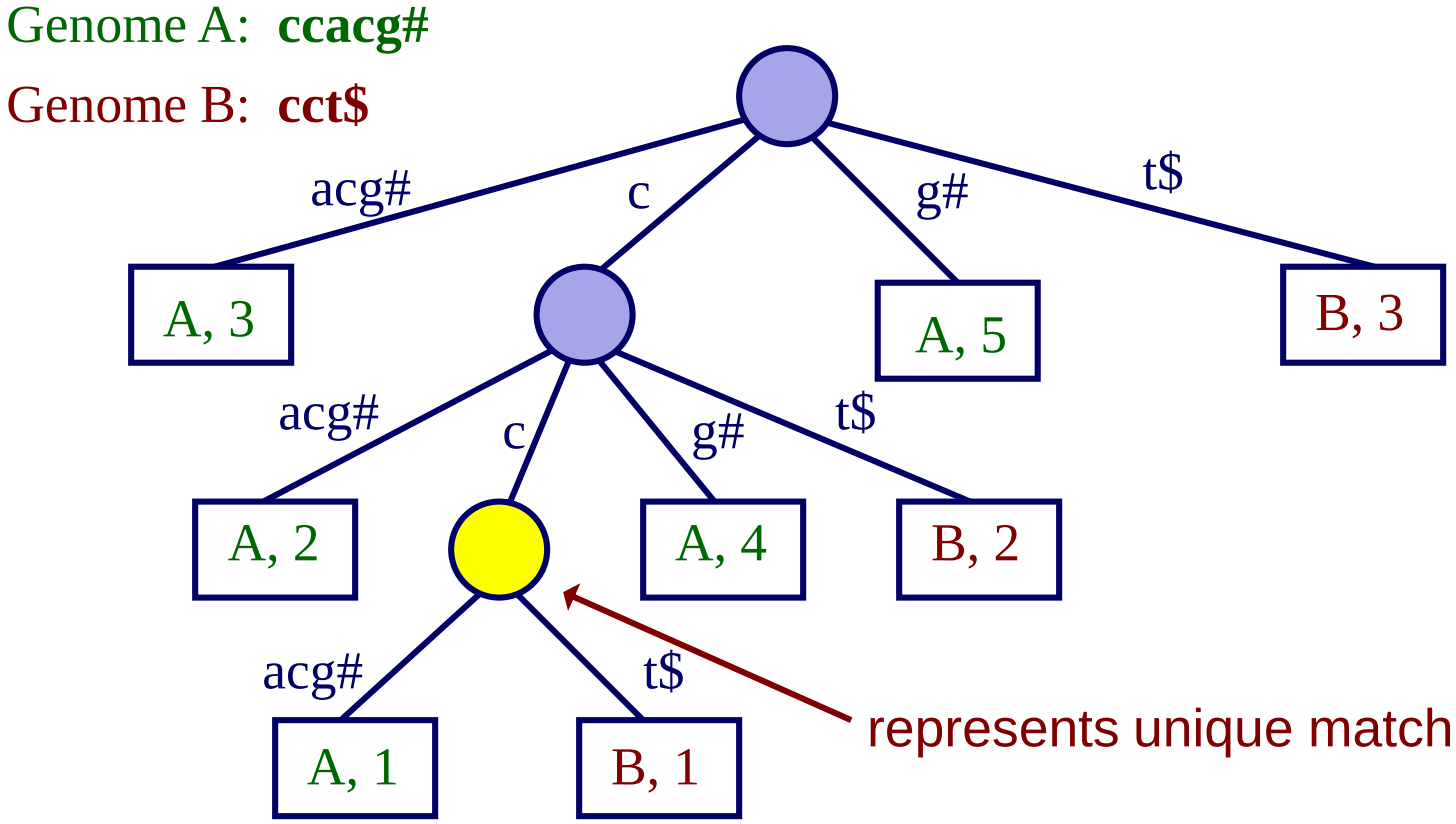
MUMs and *Generalized* Suffix Trees

- Build one suffix tree for both genomes A and B
- Label each leaf node with genome it represents



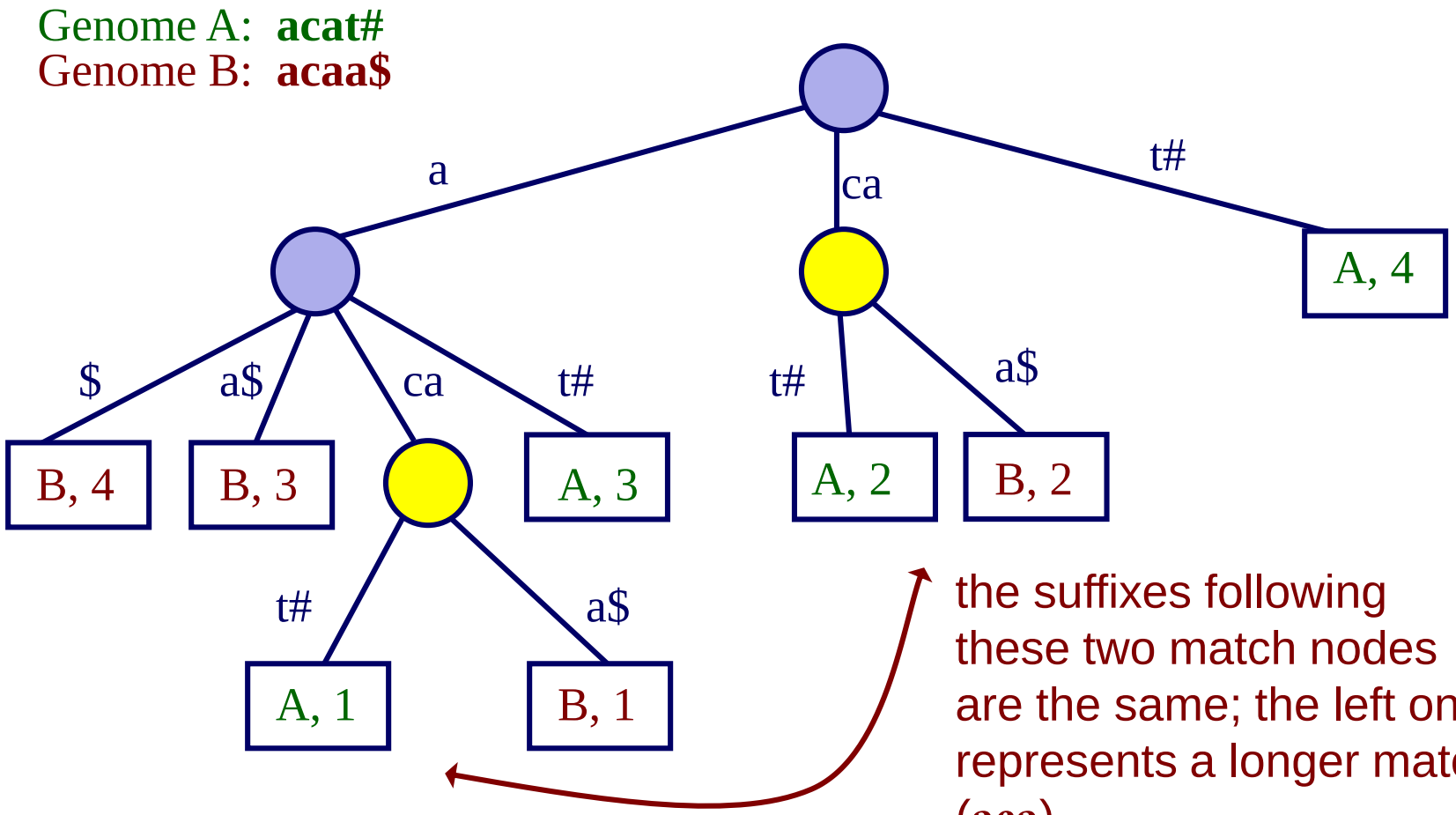
MUMs and Suffix Trees

- 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

- 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

- Main parameter of approach: length of shortest MUM that should be identified (20 – 50 bases)

Step 2: Chaining in MUMmer

- 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

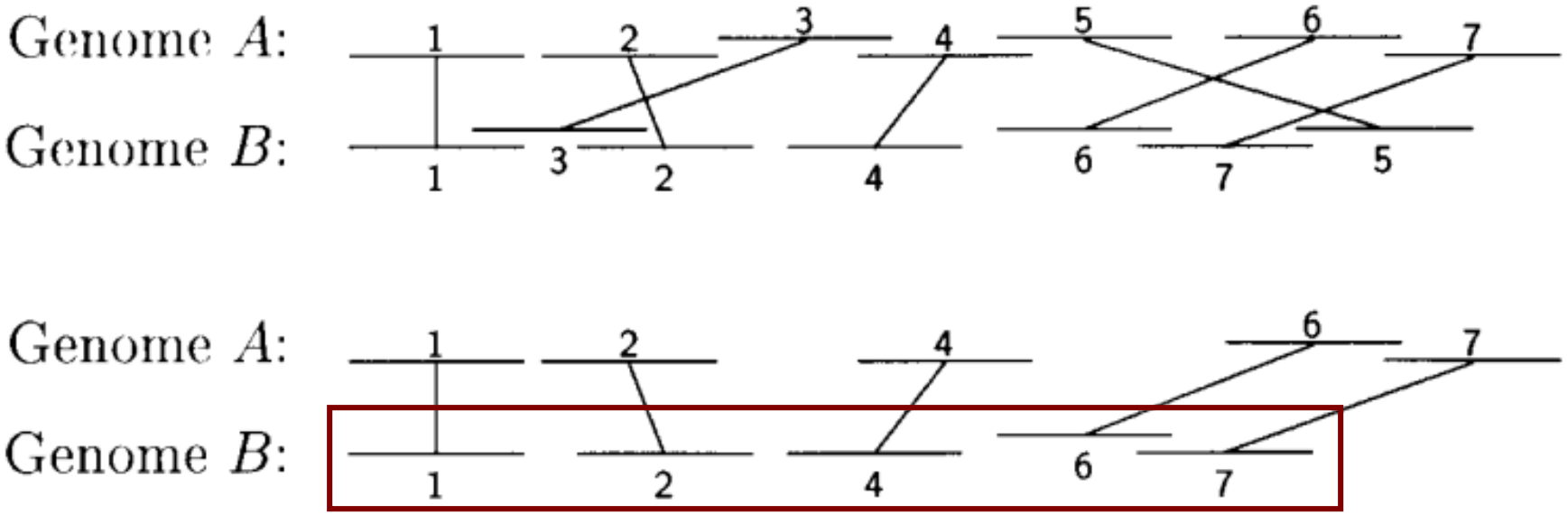
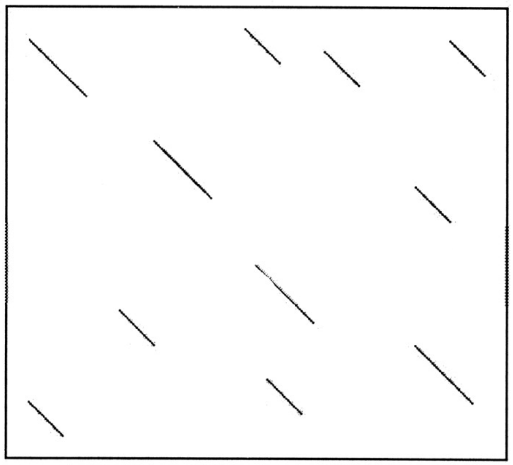


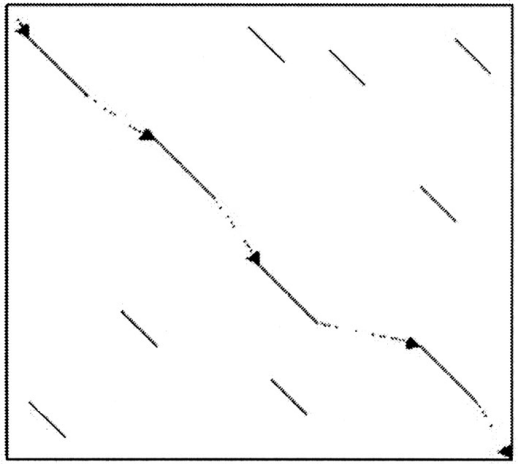
Figure from: Delcher et al., *Nucleic Acids Research* 27, 1999

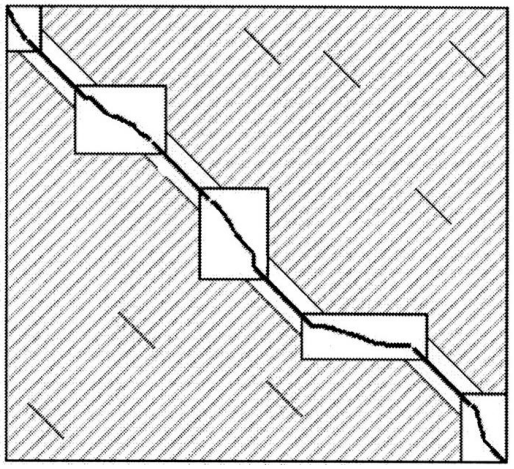
Finding Longest Subsequence

- 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

Recall: Three Main Steps of Large-Scale Alignment







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

MUMmer

1. Suffix trees to obtain MUMs
2. LIS to find colinear MUMs
3. Smith-Waterman and recursive MUMmer for gap filling

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
- Insertions
  - simple insertions: trivial to detect
  - transpositions (subsequences that were deleted from one location and inserted elsewhere): look for out-of-sequence MUMs

Step 3: Close the Gaps

- Polymorphic regions
  - short ones: align them with dynamic programming method
  - long ones: call MUMmer recursively with reduced minimum MUM length
- Repeats
  - detected by overlapping MUMs

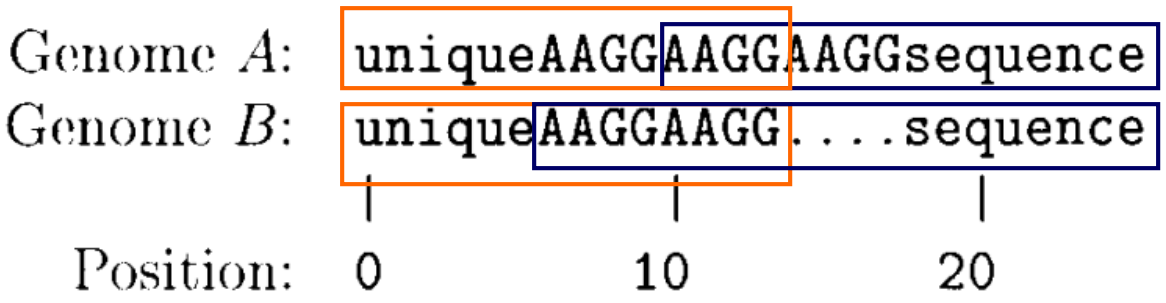


Figure from: Delcher et al. Nucleic Acids Research 27, 1999

MUMmer Performance

FASTA on 1000 base pair segments

MUMmer

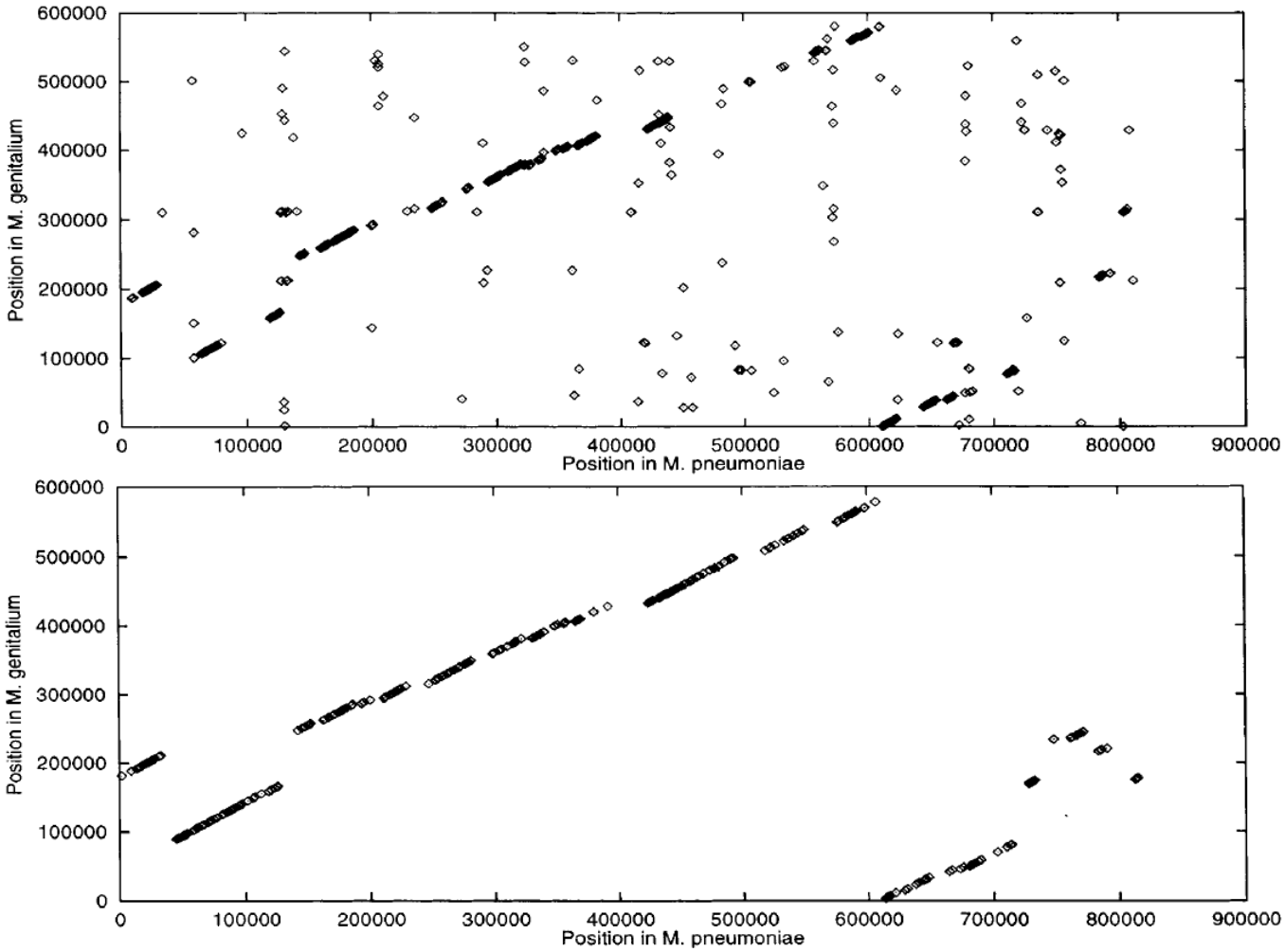


Figure from: Delcher et al. Nucleic Acids Research 27, 1999

MUMmer Performance

- *Mycoplasma* test case
- Suffix tree: 6.5s
- LIS: 0.02s
- Smith-Waterman: 116s

- FASTA baseline: many hours

DEC Alpha 4100

Centre for Computing History

Longevity of MUMmer

- Antimicrobial Resistance Identification By Assembly (ARIBA)
- Identify antimicrobial resistance genes from Illumina reads

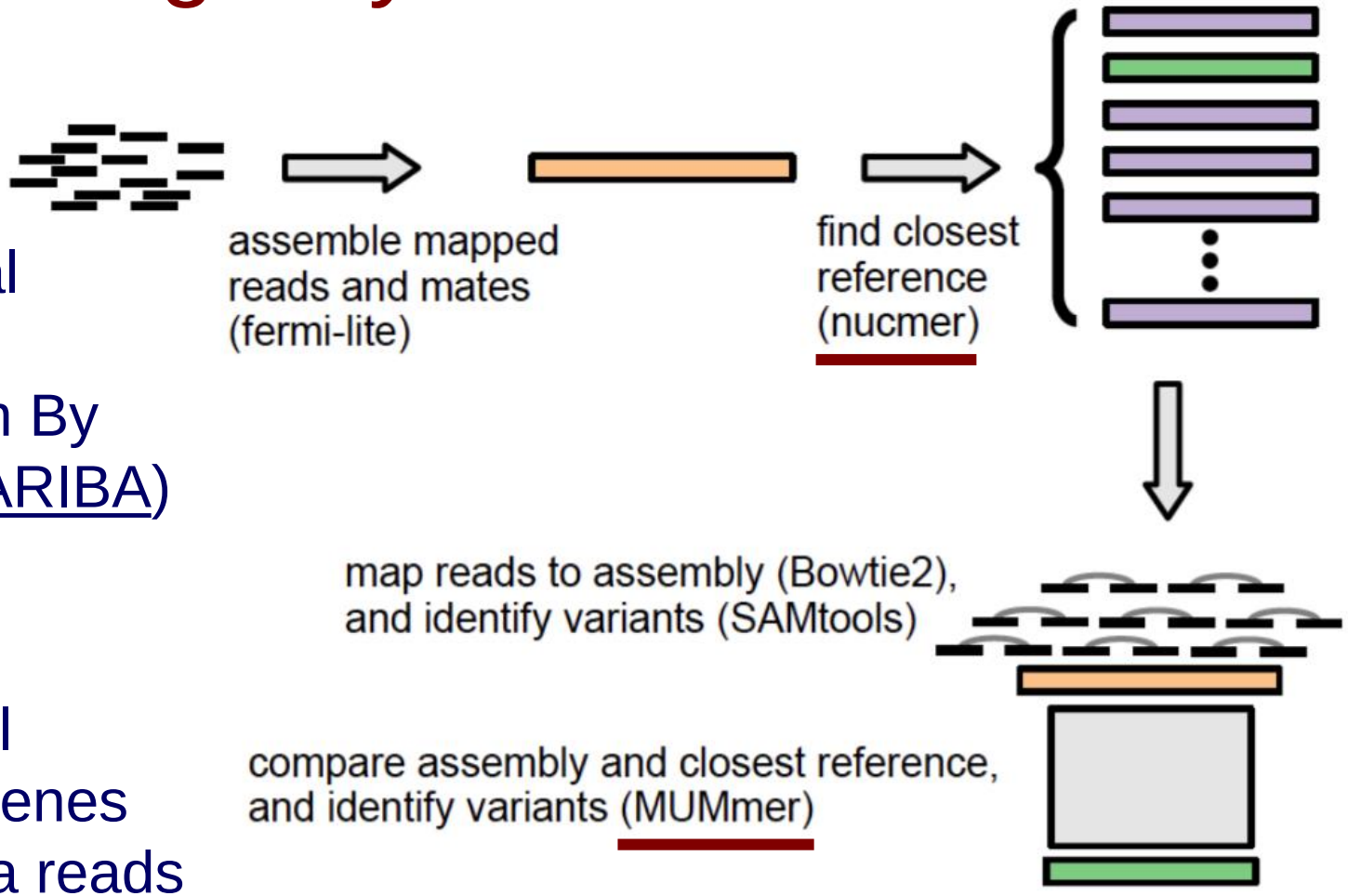


Figure from: Hunt et al. bioRxiv 2017

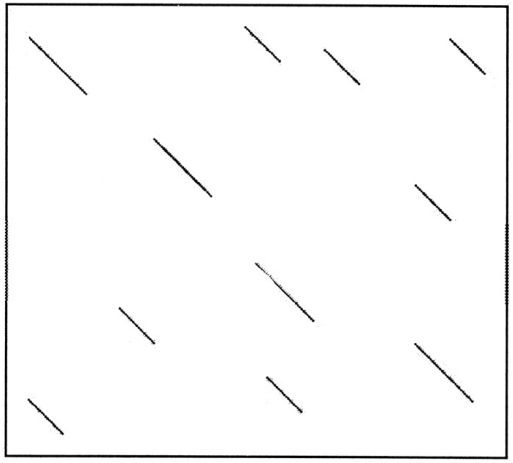
Longevity of MUMmer

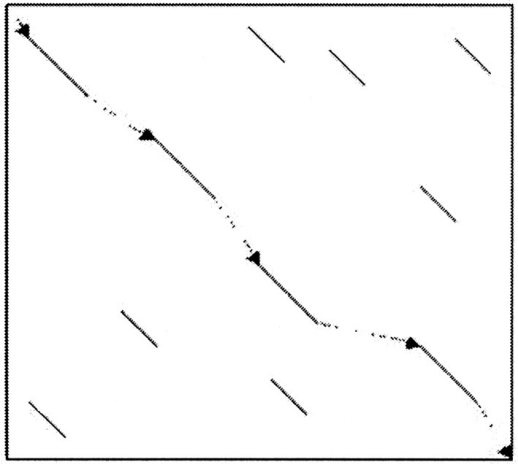
- Whole genome alignment still an active area of research
  - Jain et al. 2018 (Mashmap2): “we were able to map an error-corrected whole-genome NA12878 human assembly to the hg38 human reference genome in about **one minute total execution time** and **< 4 GB memory** using 8 CPU threads”
  - Uses MUMmer as ground truth in evaluation

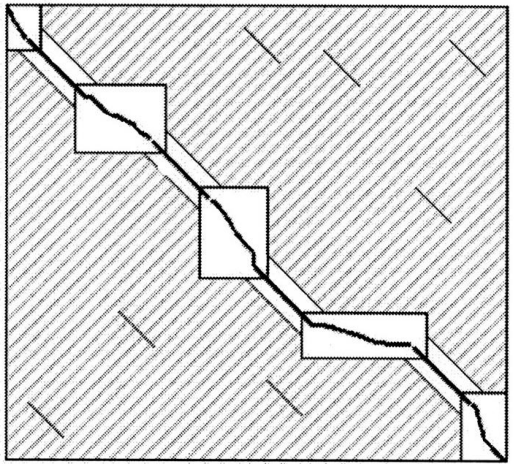
Limitations of MUMmer

- MUMs are perfect matches, typically ≥ 20-50 base pairs
- Evolutionarily distant may not have sufficient MUMs to anchor global alignment
- How can we tolerate minor variation in the seeds?

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

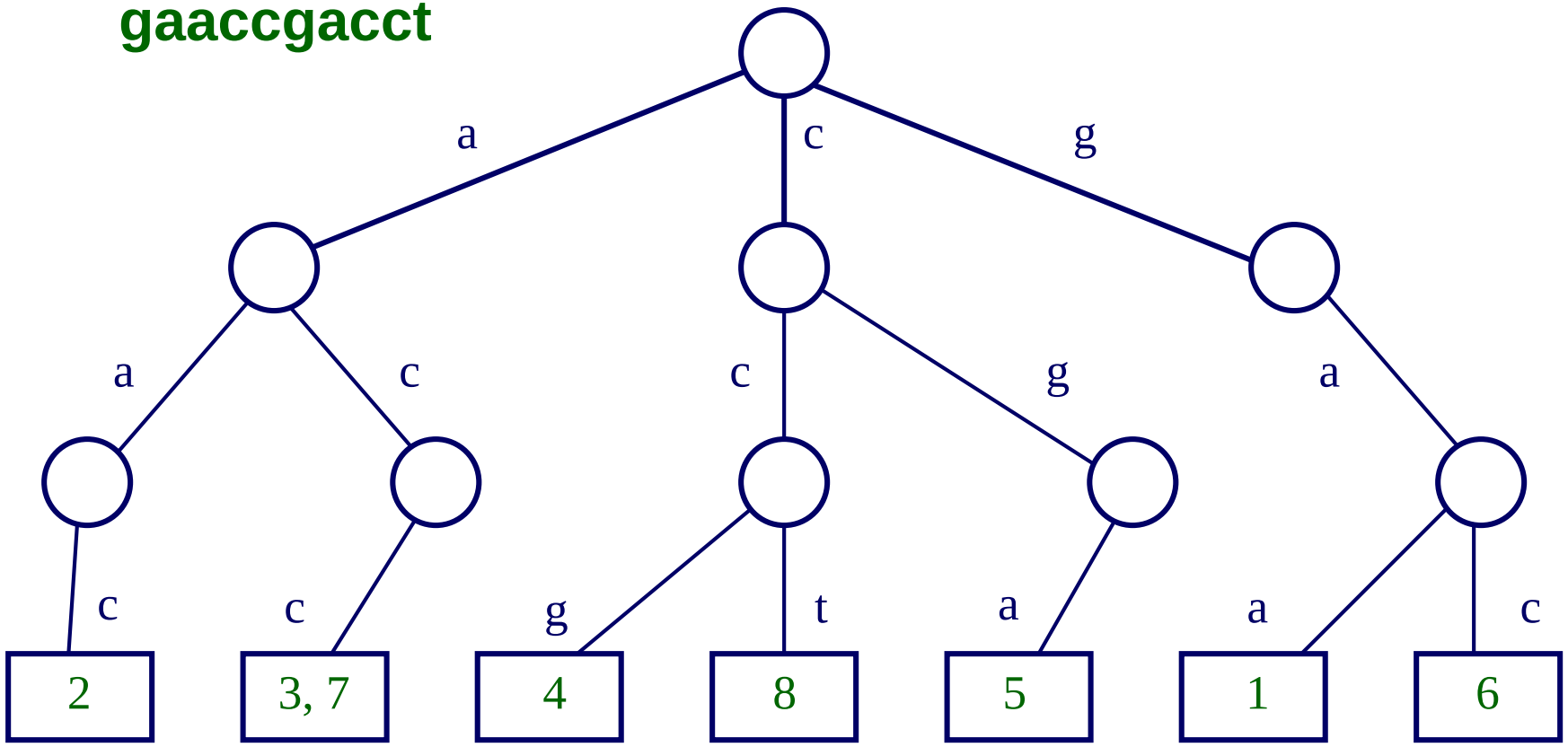
Step 1: Finding Seeds in LAGAN

- *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

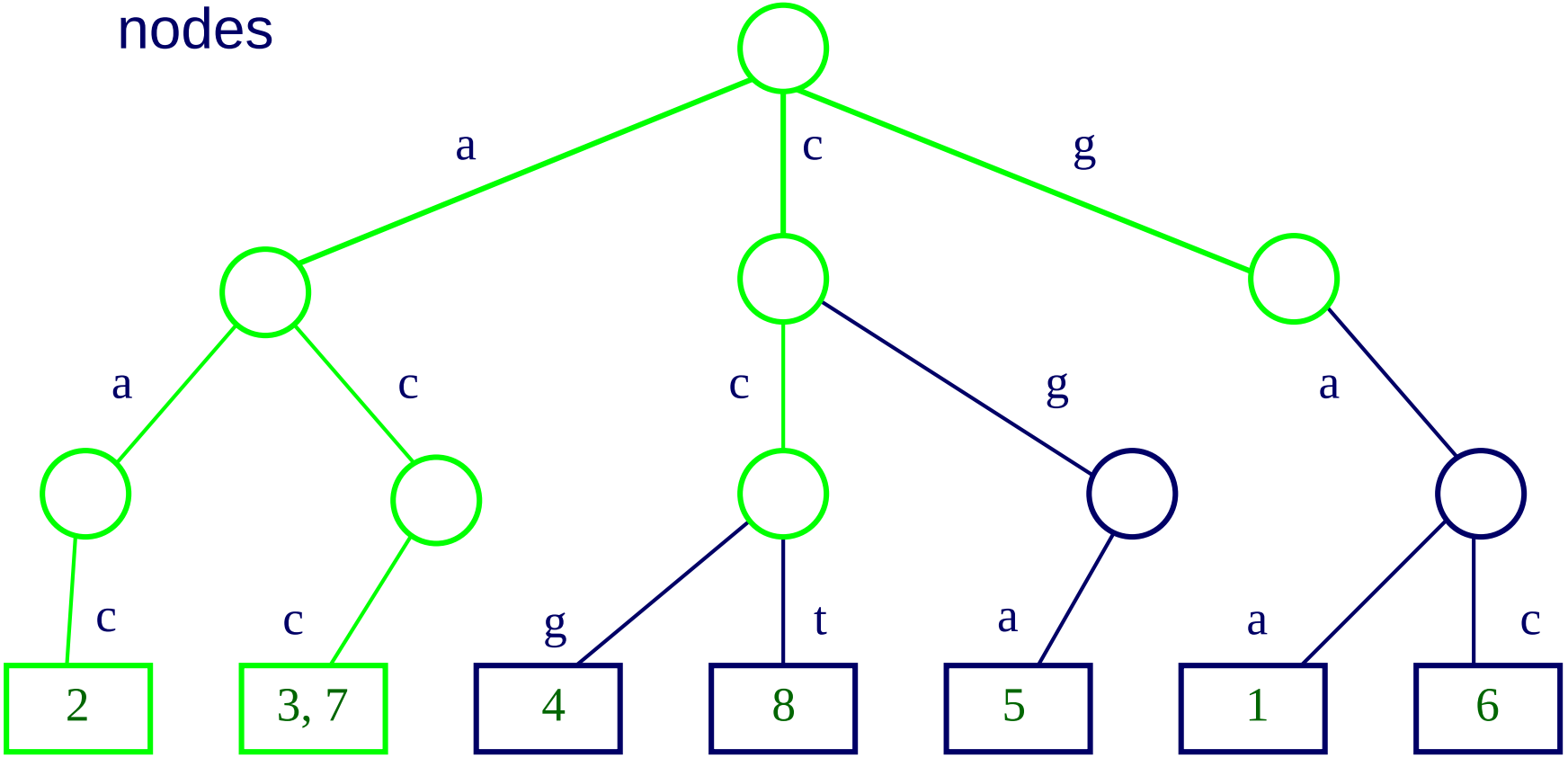
- Example: a *trie* to represent all 3-mers of the sequence **gaaccgacct**



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

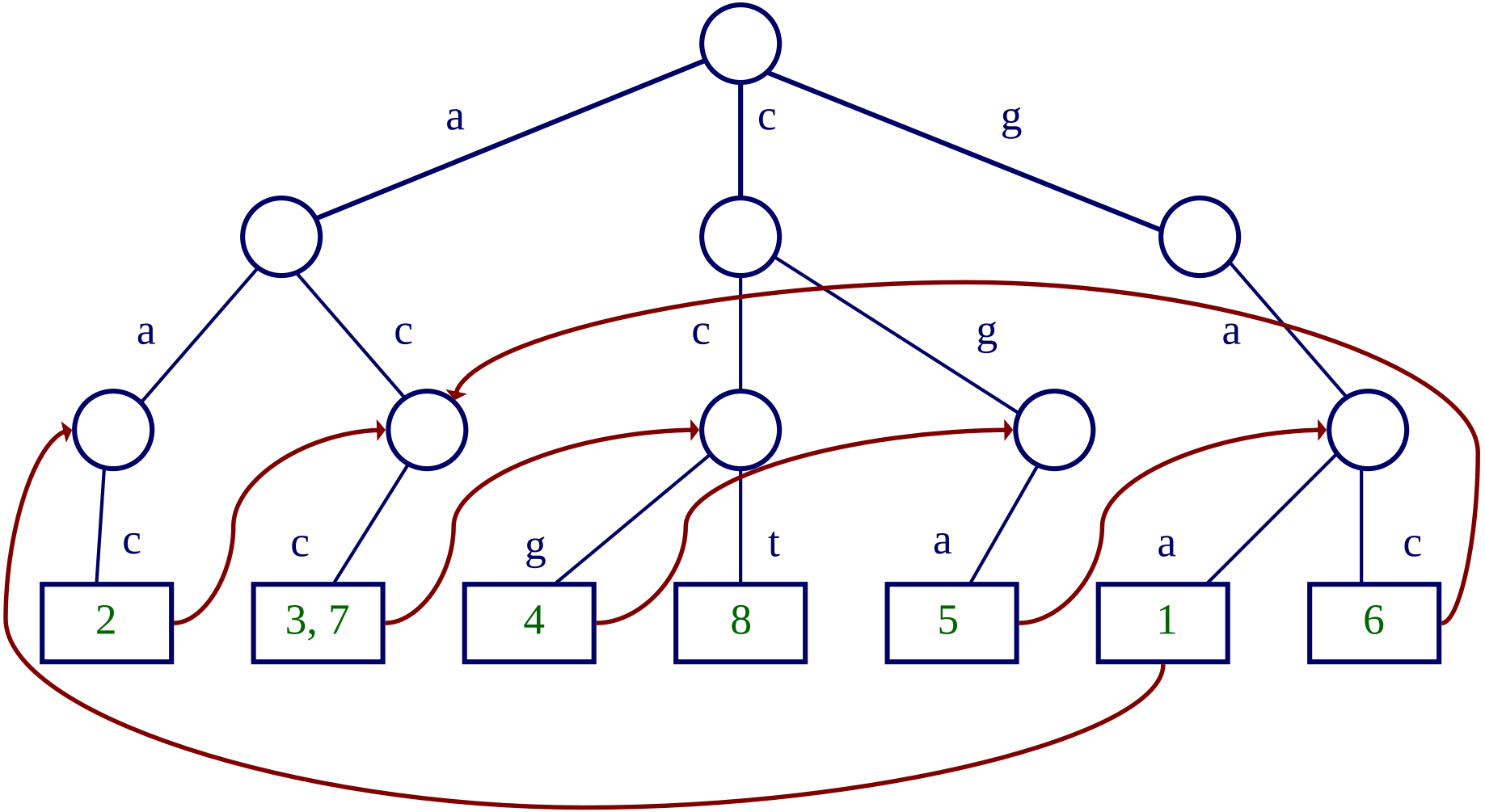
Allowing Degenerate Matches

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



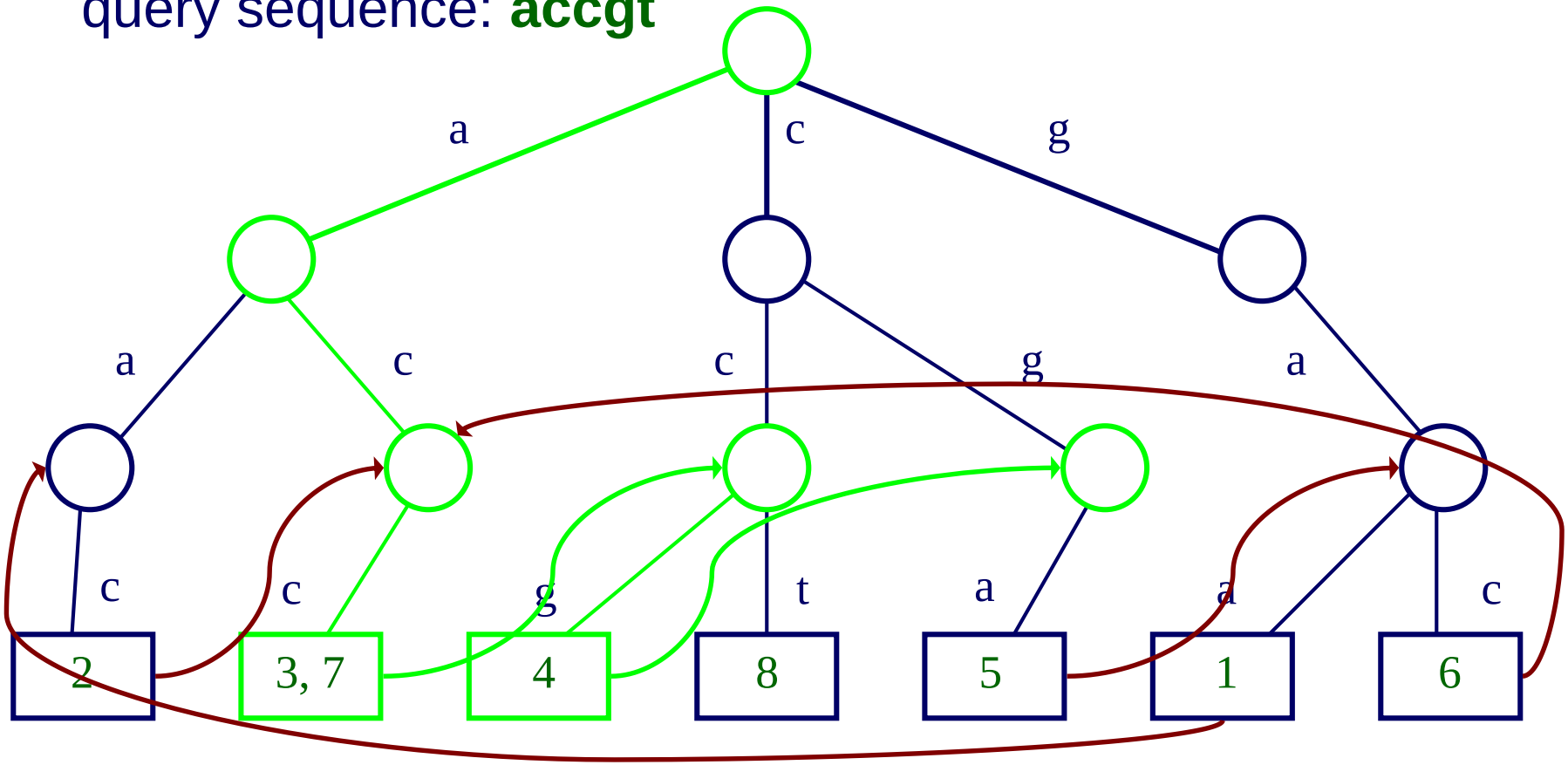
LAGAN Uses Threaded Tries

- 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**

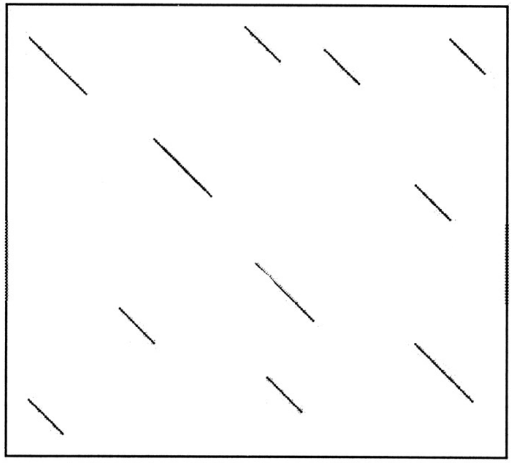


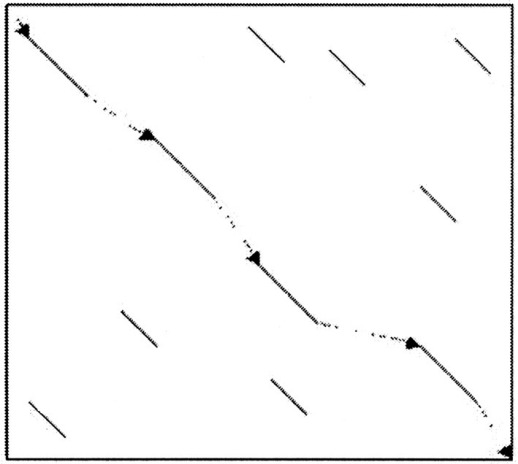
- Usually requires following only two pointers to match against the next *k*-mer, instead of traversing tree from root for each

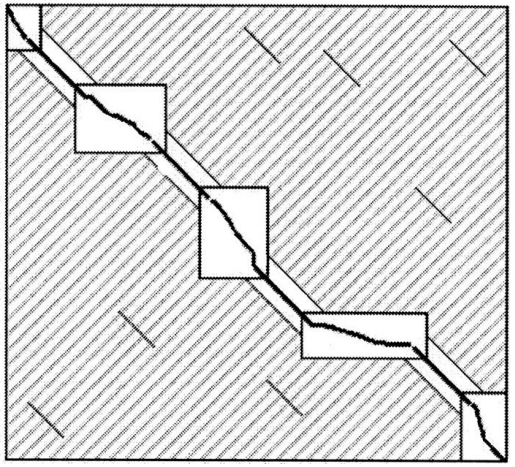
Comparing MUMmer and LAGAN

	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.	2.	3.
MUMmer	Pattern matching to find seeds for global alignment	Find a good chain of anchors	Fill in with standard but constrained alignment
LAGAN	Suffix trees to obtain MUMs	Longest Increasing Subsequence	Smith-Waterman, recursive MUMmer
	k-mer trie to obtain seeds	Spare dynamic programming	Dynamic programming

Multiple Whole Genome Alignment: Task Definition

Given

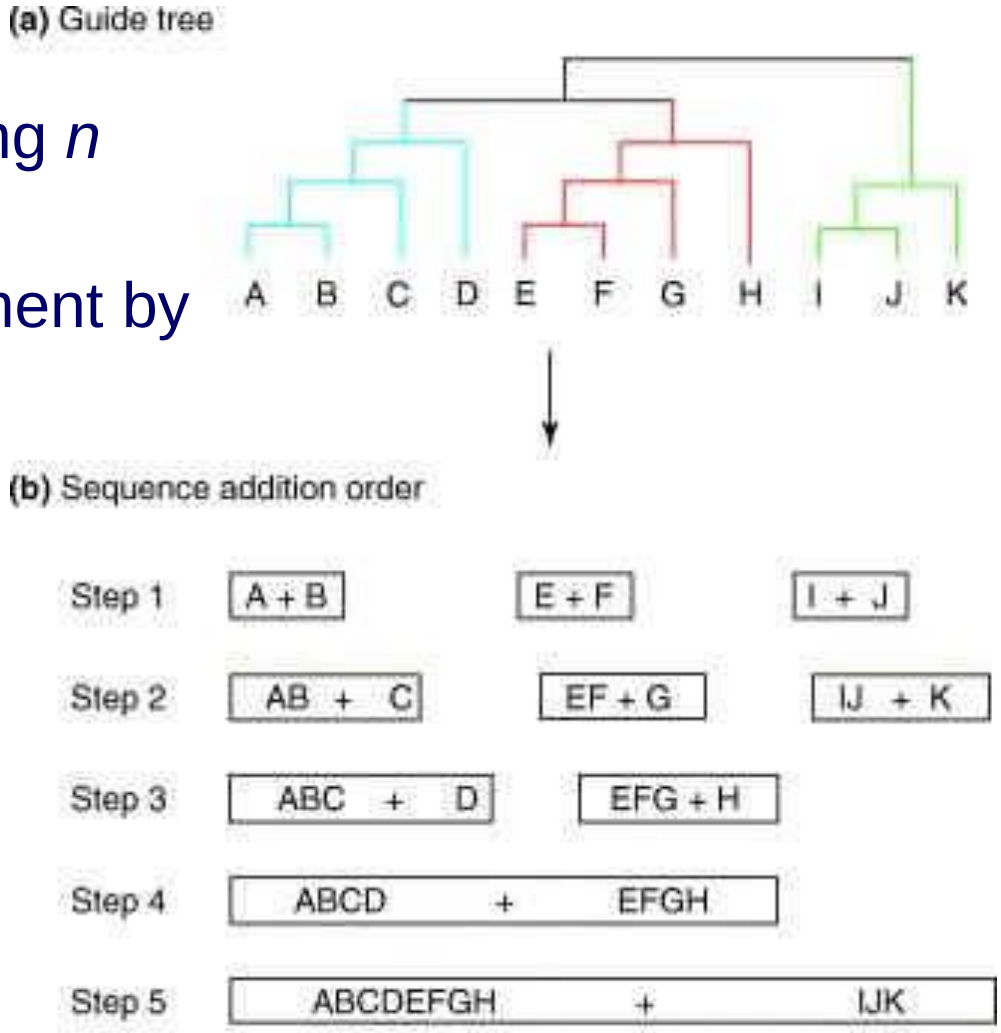
- A set of $n > 2$ genomes (or other large-scale sequences)

Do

- Identify all corresponding positions between all genomes, allowing for substitutions, insertions/deletions, and *rearrangements*

Progressive Alignment

- Given a *guide tree* relating *n* genomes
- Construct multiple alignment by performing *n*-1 pairwise alignments

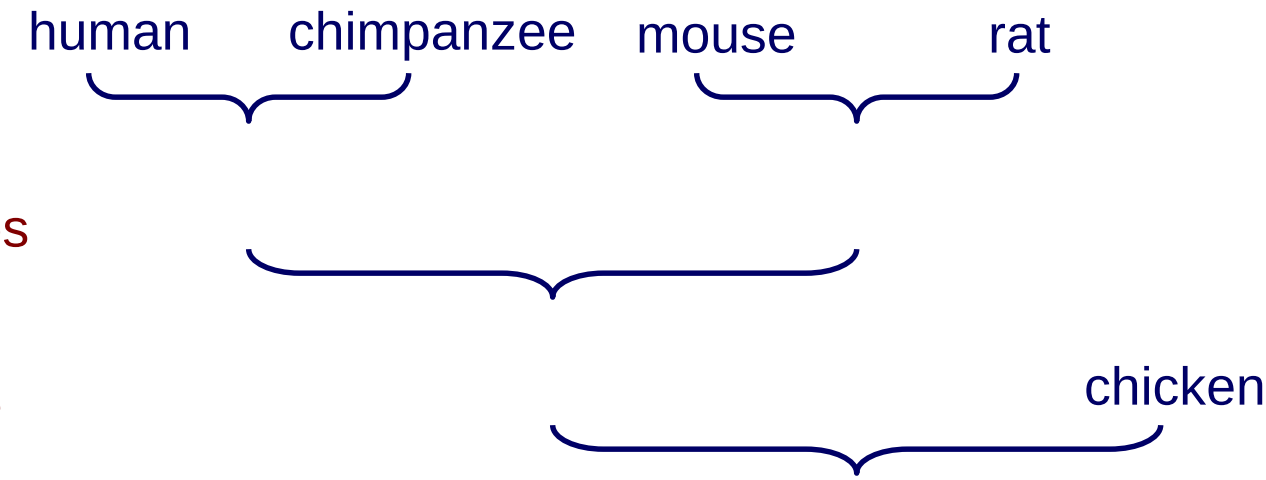


Progressive Alignment: MLAGAN Example

align pairs of sequences

align multi-sequences (alignments)

align multi-sequence with sequence



Progressive Alignment: MLAGAN Example

Suppose we're aligning the multi-sequence X/Y with Z

1. anchors from X-Z and Y-Z become anchors for X/Y-Z
2. overlapping anchors are reweighted
3. LIS algorithm is used to chain anchors

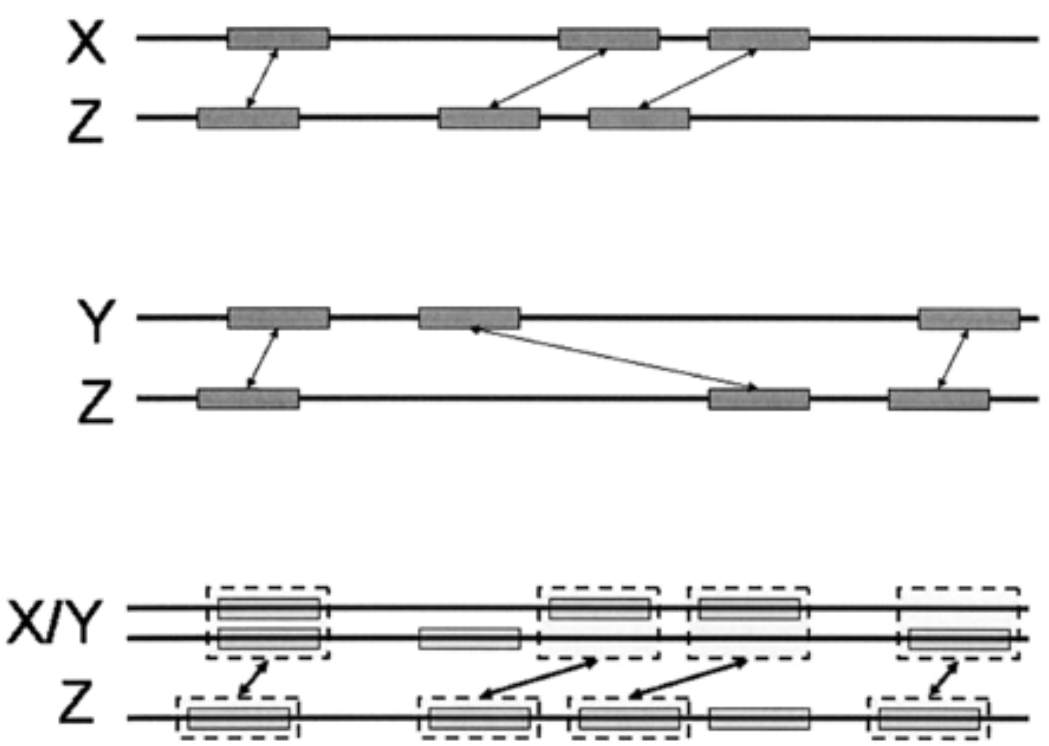
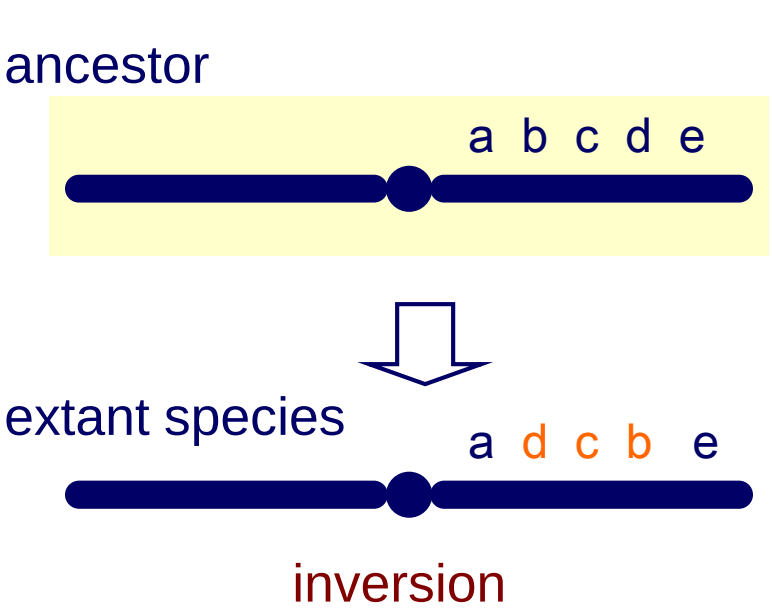
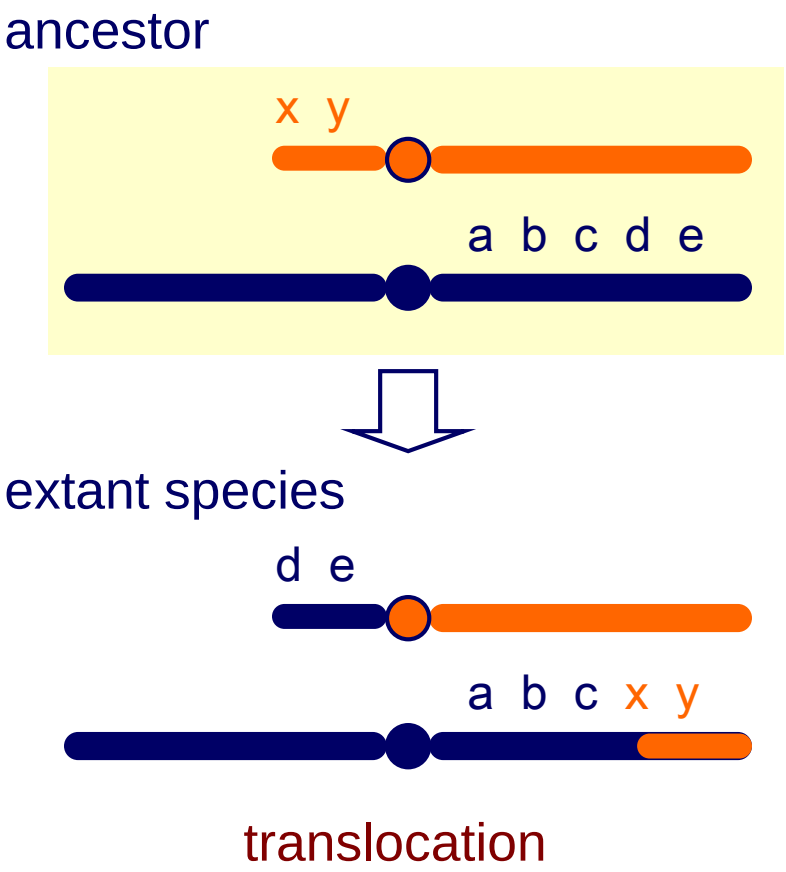


Figure from: Brudno et al. *Genome Research*, 2003

Genome Rearrangements



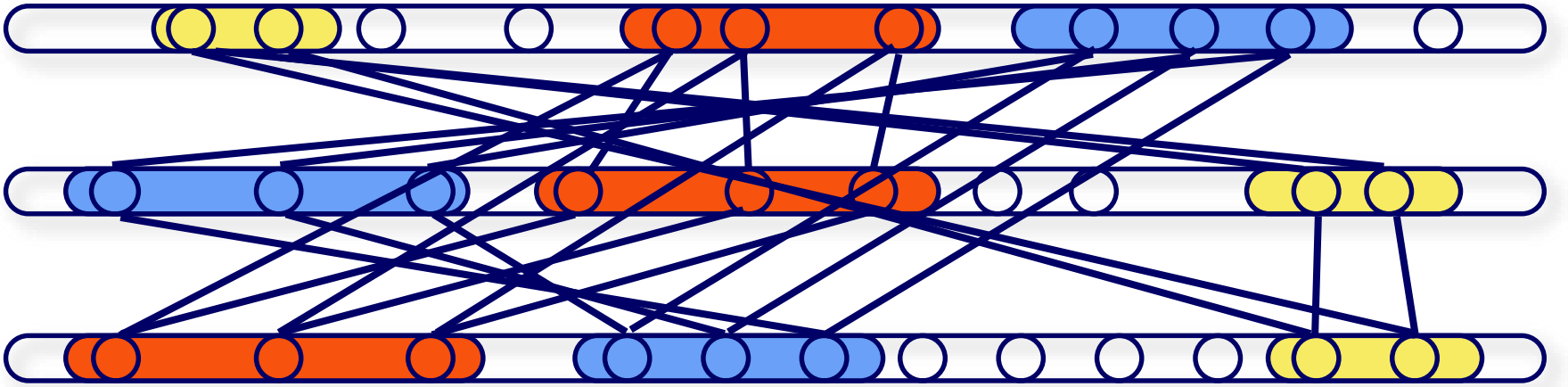


- Can occur within a chromosome or across chromosomes
- Can have combinations of these events

Mercator: Rough Orthology Map

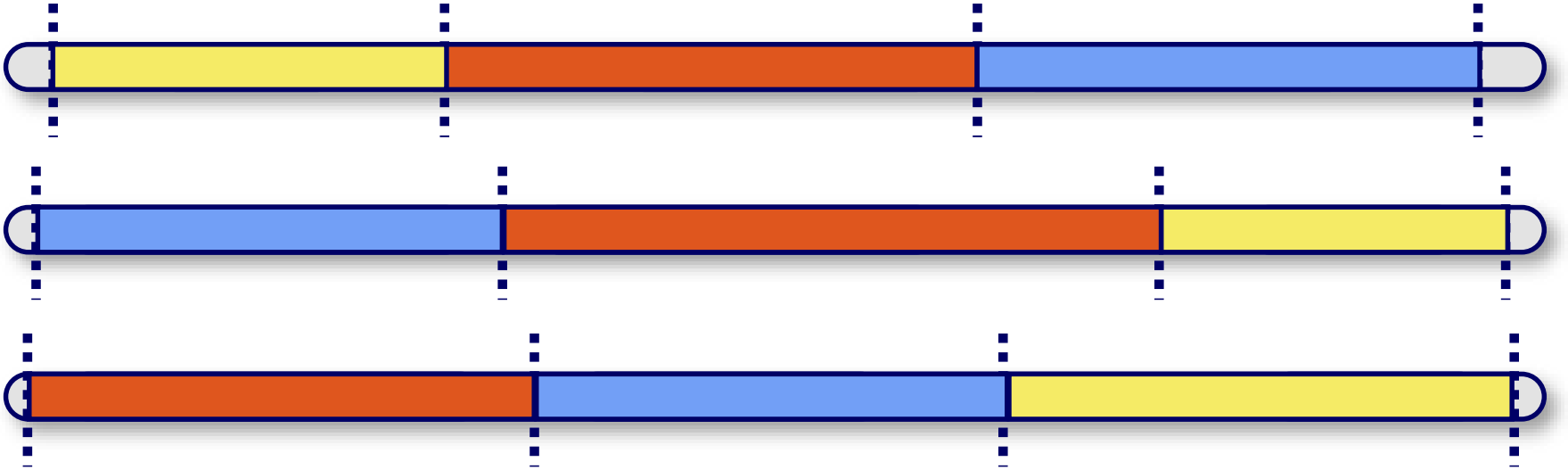
k-partite graph with edge weights

vertices = anchors, edges = sequence similarity



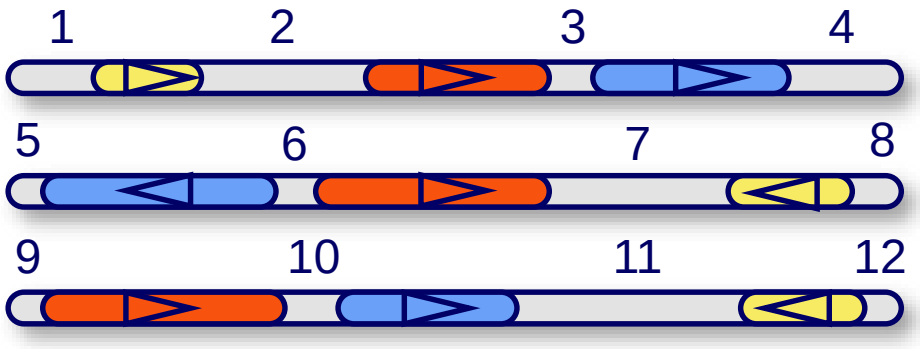
Refining the Map: Finding Breakpoints

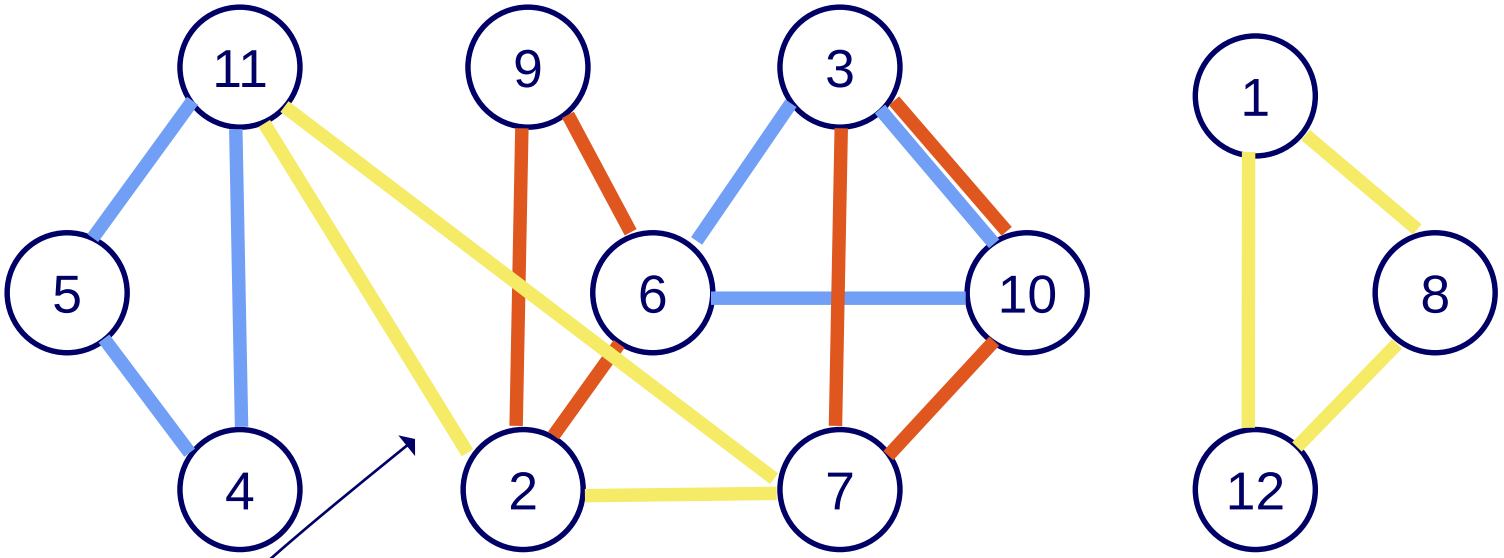
- *Breakpoints*: the positions at which genomic rearrangements disrupt colinearity of segments



- Mercator finds breakpoints by using inference in an *undirected graphical model*

The Breakpoint Graph





some prefix of region 2 and some prefix of region 11 should be aligned

Comparing MLAGAN and Mercator

- MLAGAN
  - Requires phylogenetic tree
  - Greedy solution with local refinement

- Mercator
  - Define probabilistic model to solve globally
  - Inference is intractable, resort to approximations