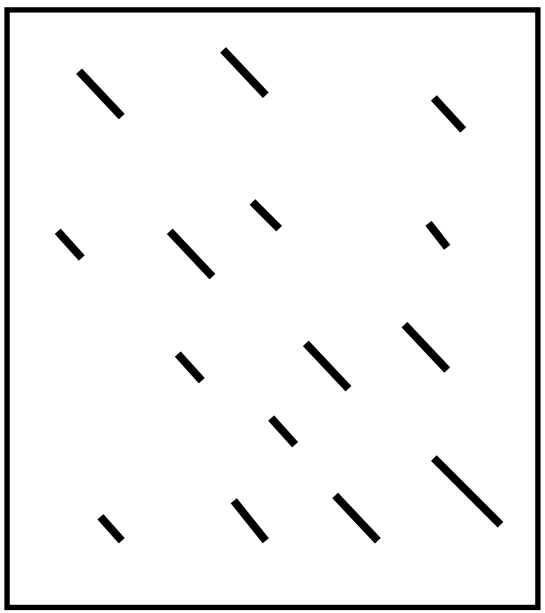
BMI/CS 776
Lecture #19
Alignment of long sequences

Colin Dewey
April 1, 2008

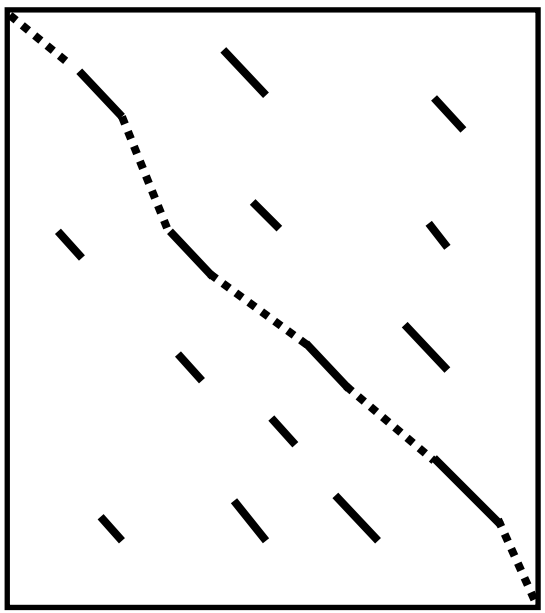
Difficulty of long sequences

- In general, long sequences are less likely to be colinear due to rearrangements
  - Today - assume colinearity
- Standard pairwise alignment is $O(n^2)$
  - Needleman-Wunsch
  - Pair HMMs

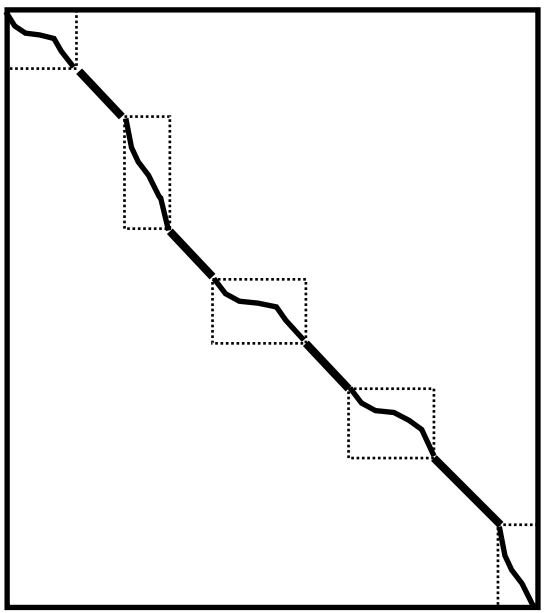
Basic strategy



Perform pattern matching to find promising local alignments (*anchors*)



Find good *chain* of *anchors*



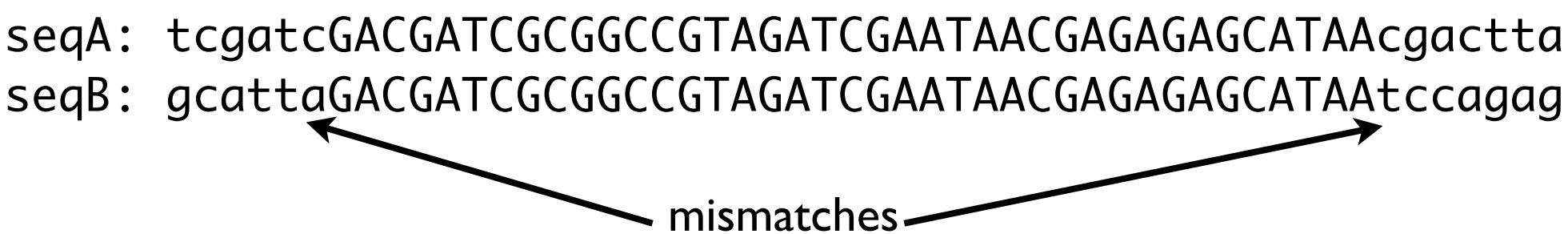
Perform standard alignment in between anchors in chain

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

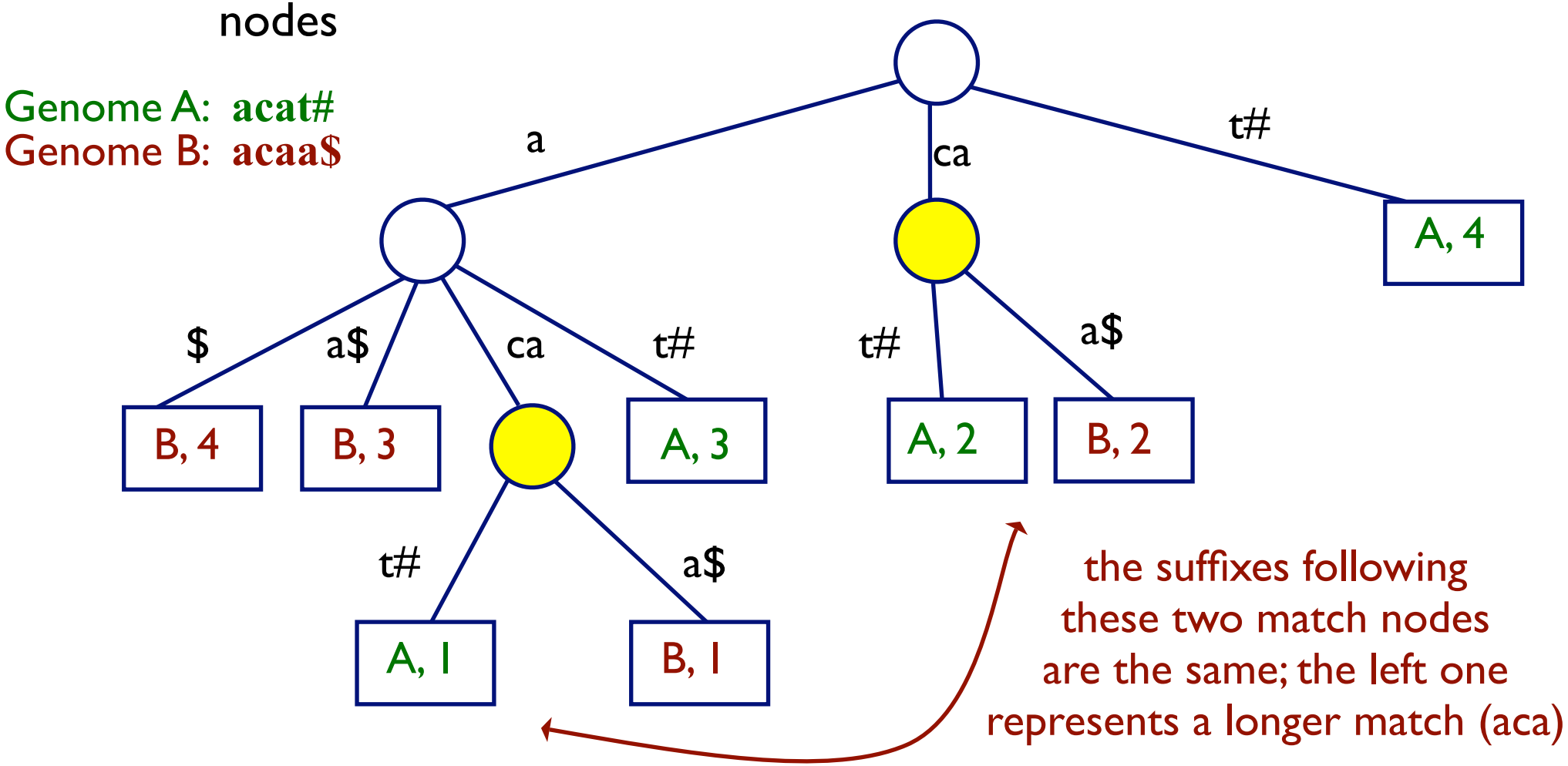
MUMmer anchors

- MUM = Maximal Unique Match
  - Match: Exact matching k-mer
  - Maximal: Not contained within another MUM
  - Unique: k-mer occurs exactly once in each sequence
- Long MUMs likely to be part of alignment



Finding MUMs with suffix trees

- unique match: internal node with 2 children, leaf nodes from different genomes
- check for maximal: compare suffixes following unique match nodes



MUM complexity

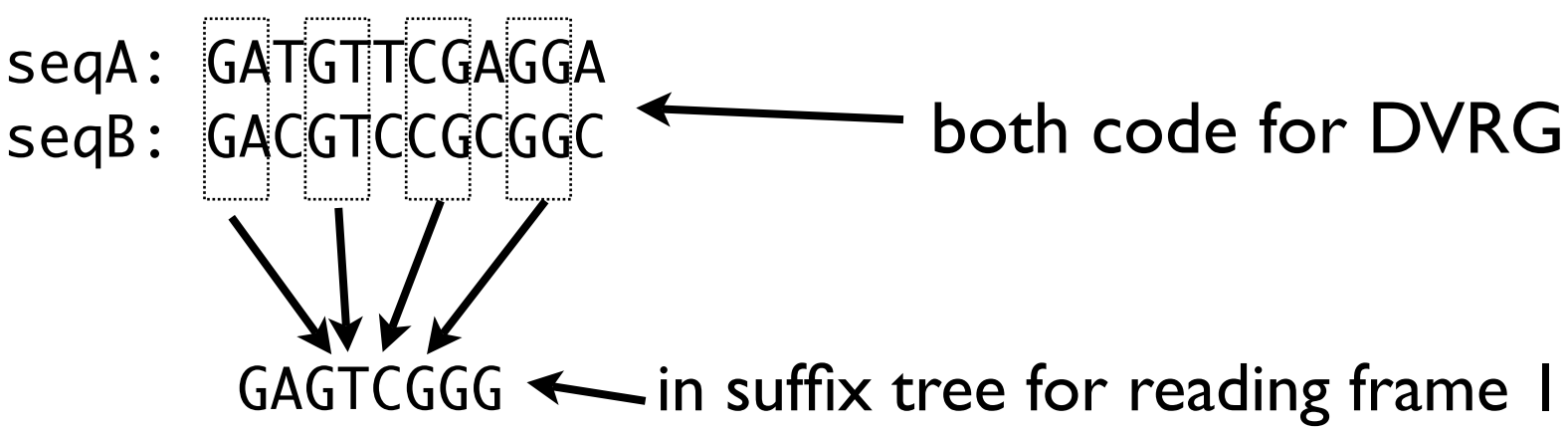
- O(n) time to construct suffix tree for both sequences (of lengths < n)
- O(n) time to find MUMs - one scan of the tree (which is O(n) in size)
- O(n) possible MUMs in contrast to $O(n^2)$ possible exact matches

AVID anchors

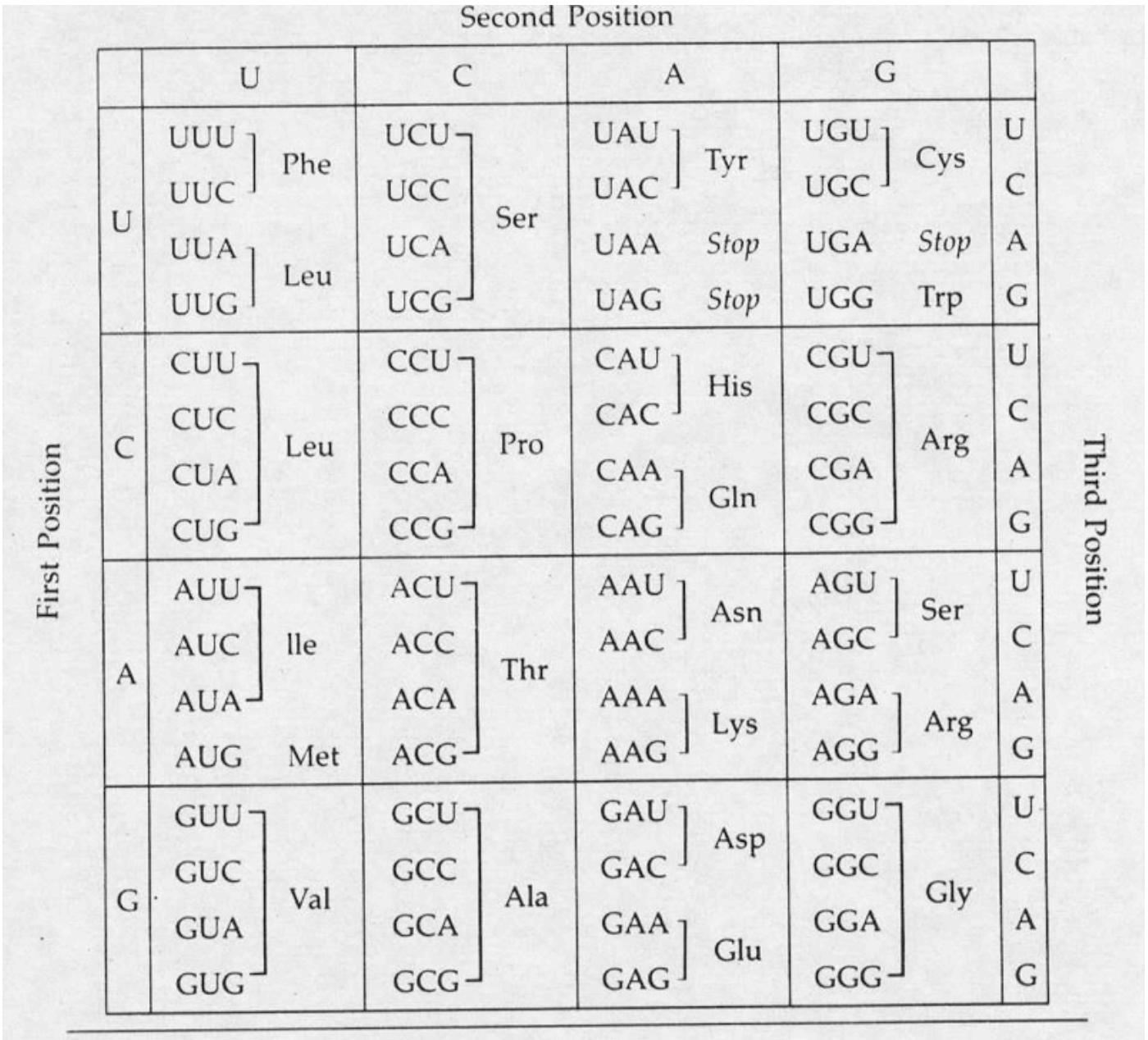
- All maximal exact matches > some minimum length
  - Suffix tree construction + traversal
- Divide matches into "clean" or "repeat" depending on whether intervals overlap a repetitive element (annotated by RepeatMasker)
  - repeat matches used only after all clean matches are considered
- Also locate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

Wobble matches

- Trick for better alignment of protein-coding DNA
- Substitutions in 3rd codon position often do not change amino acid
- Look for exact matches ignoring every 3rd base
- Build suffix tree for all 3 reading frames



Codon table



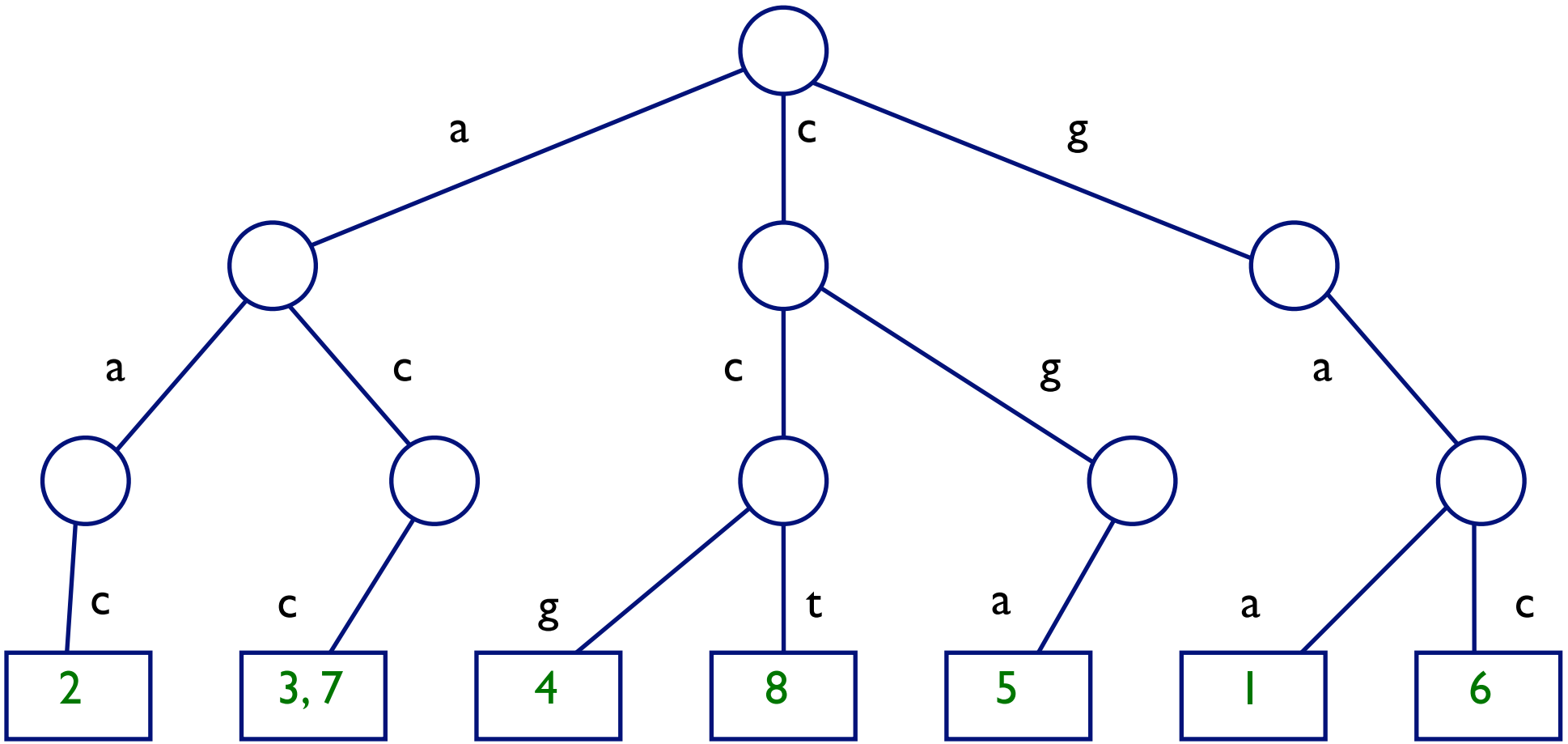
First Position	Second Position: U	Second Position: C	Second Position: A	Second Position: G	Third Position
U	UUU Phe	UCU Ser	UAU Tyr	UGU Cys	U
	UUC Phe	UCC Ser	UAC Tyr	UGC Cys	C
	UUA Leu	UCA Ser	UAA *Stop*	UGA *Stop*	A
	UUG Leu	UCG Ser	UAG *Stop*	UGG Trp	G
C	CUU Leu	CCU Pro	CAU His	CGU Arg	U
	CUC Leu	CCC Pro	CAC His	CGC Arg	C
	CUA Leu	CCA Pro	CAA Gln	CGA Arg	A
	CUG Leu	CCG Pro	CAG Gln	CGG Arg	G
A	AUU Ile	ACU Thr	AAU Asn	AGU Ser	U
	AUC Ile	ACC Thr	AAC Asn	AGC Ser	C
	AUA Ile	ACA Thr	AAA Lys	AGA Arg	A
	AUG Met	ACG Thr	AAG Lys	AGG Arg	G
G	GUU Val	GCU Ala	GAU Asp	GGU Gly	U
	GUC Val	GCC Ala	GAC Asp	GGC Gly	C
	GUA Val	GCA Ala	GAA Glu	GGA Gly	A
	GUG Val	GCG Ala	GAG Glu	GGG Gly	G

LAGAN anchors

- Uses inexact, gapped matches (local alignments) as anchors
- CHAOS: finds local alignments
  - threaded trie for inexact k-mer matching
  - chaining of k-mers

LAGAN - Using *Tries* to Find Seeds

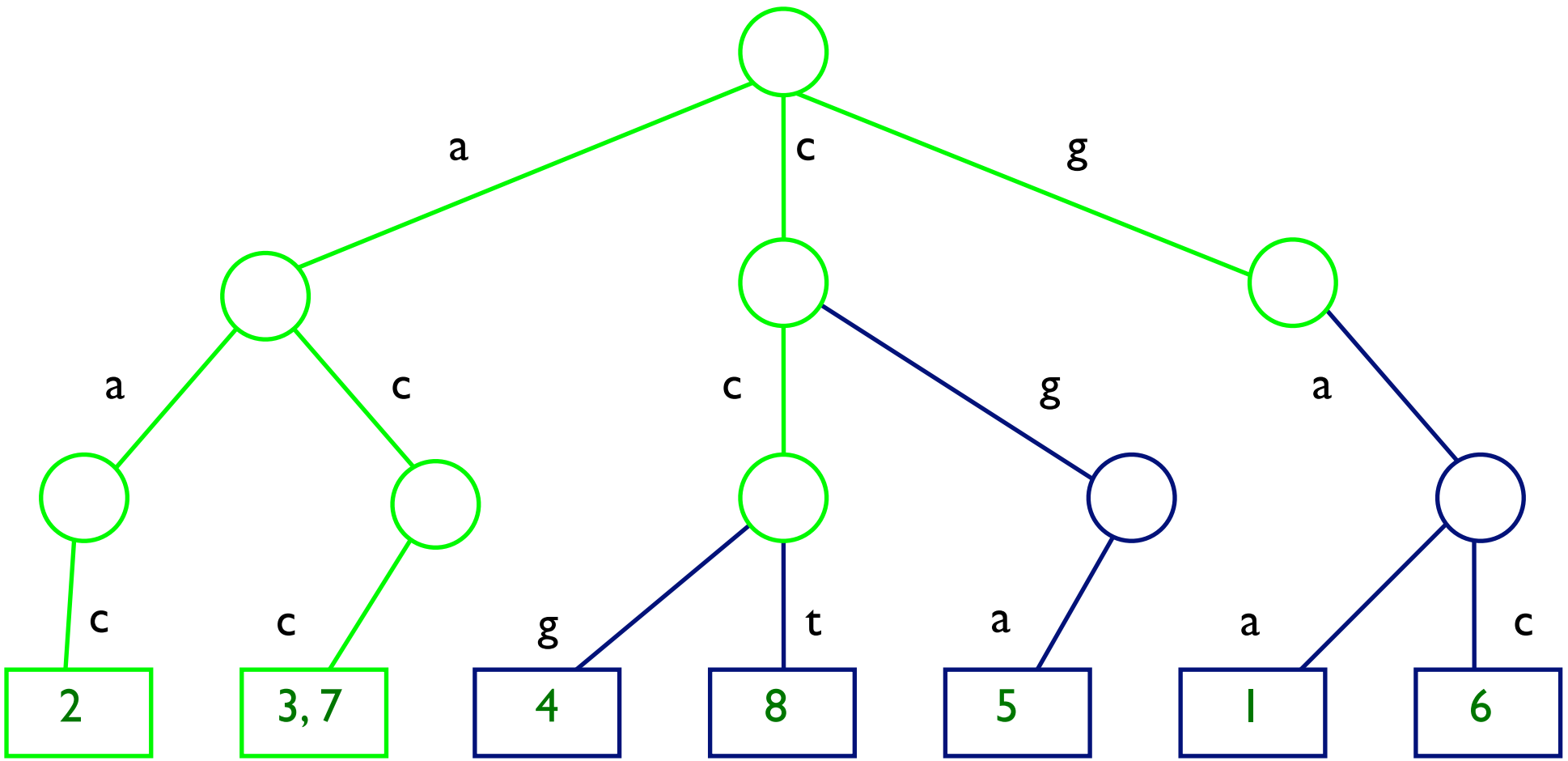
- a *trie* to represent all 3-mers of the sequence **gaaccgacct**



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

Allowing Degenerate Matches

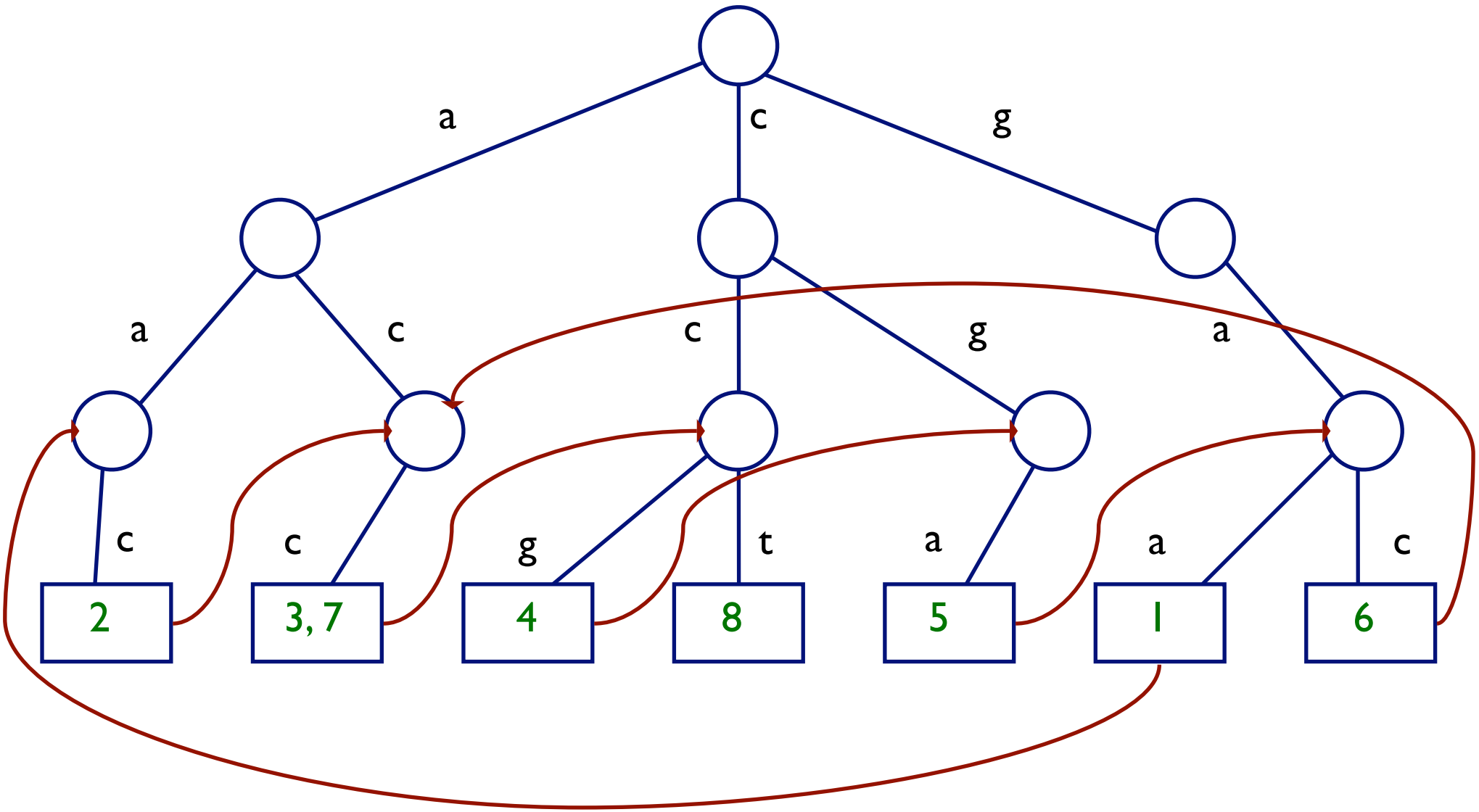
- suppose we're allowing 1 base to mismatch in looking for matches to the 3-mer **acc**; need to explore green nodes



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

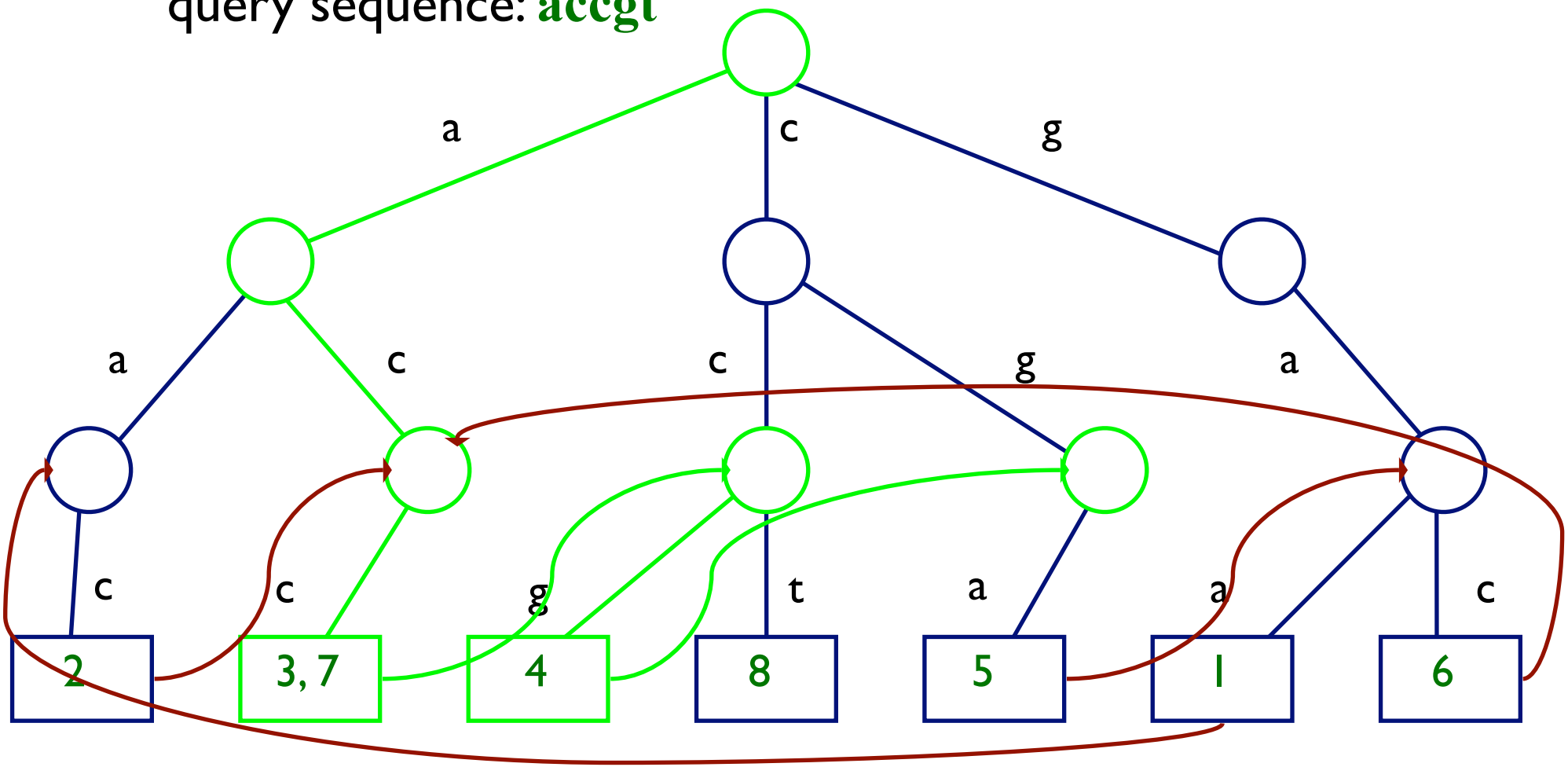
LAGAN Uses Threaded Tries

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



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

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

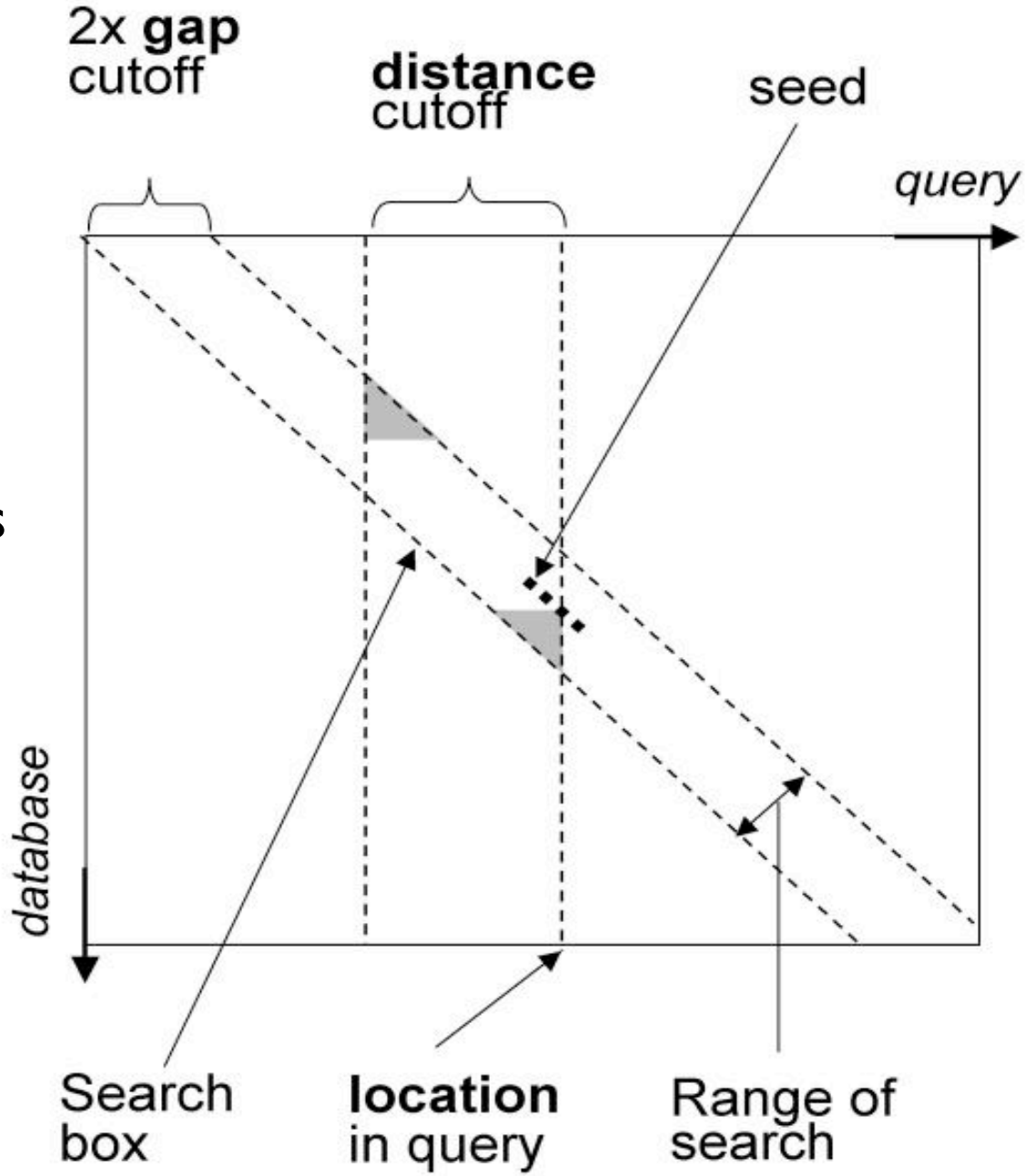


Figure from: Brudno et al. *BMC Bioinformatics*, 2003

Longest Increasing Subsequence

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

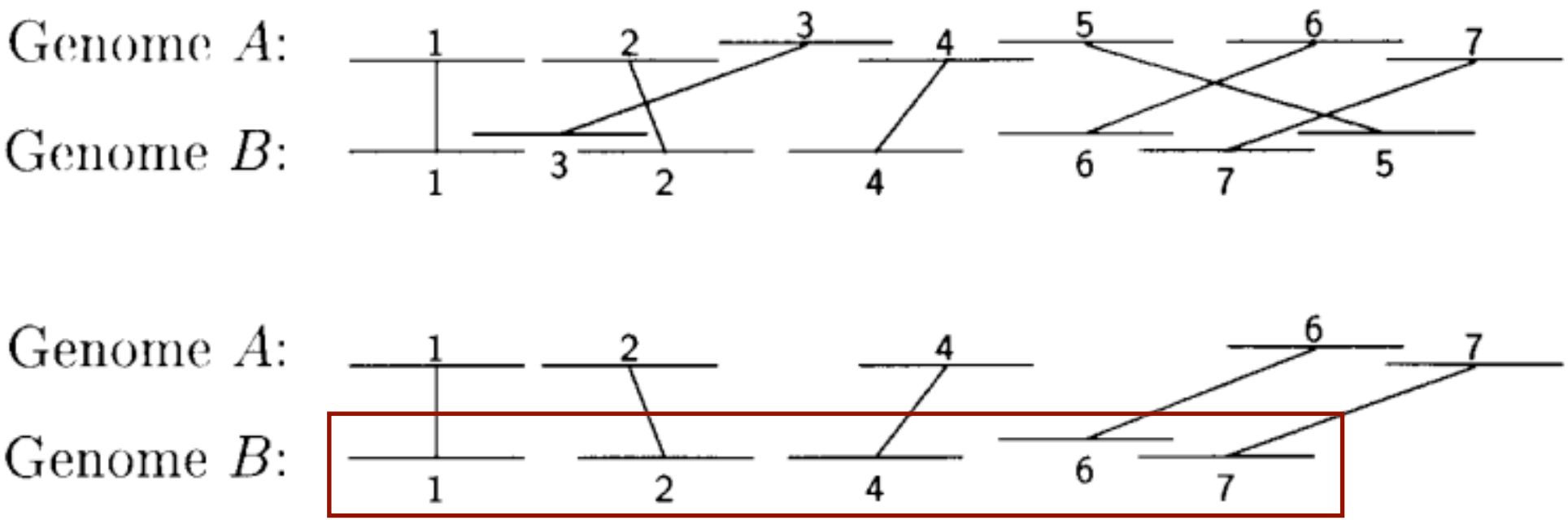


Figure from: Delcher et al., *Nucleic Acids Research* 27, 1999

Chaining anchors 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 = -∞
  - Match score = score of local alignment around anchor

Sparse Dynamic Programming

- Eppstein et al., 1992a,b
  - Find best chain of non-overlapping local alignments
  - With gap scores, match scores
  - O(nlogn), where n is the number of local alignments

LAGAN final alignment

- 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

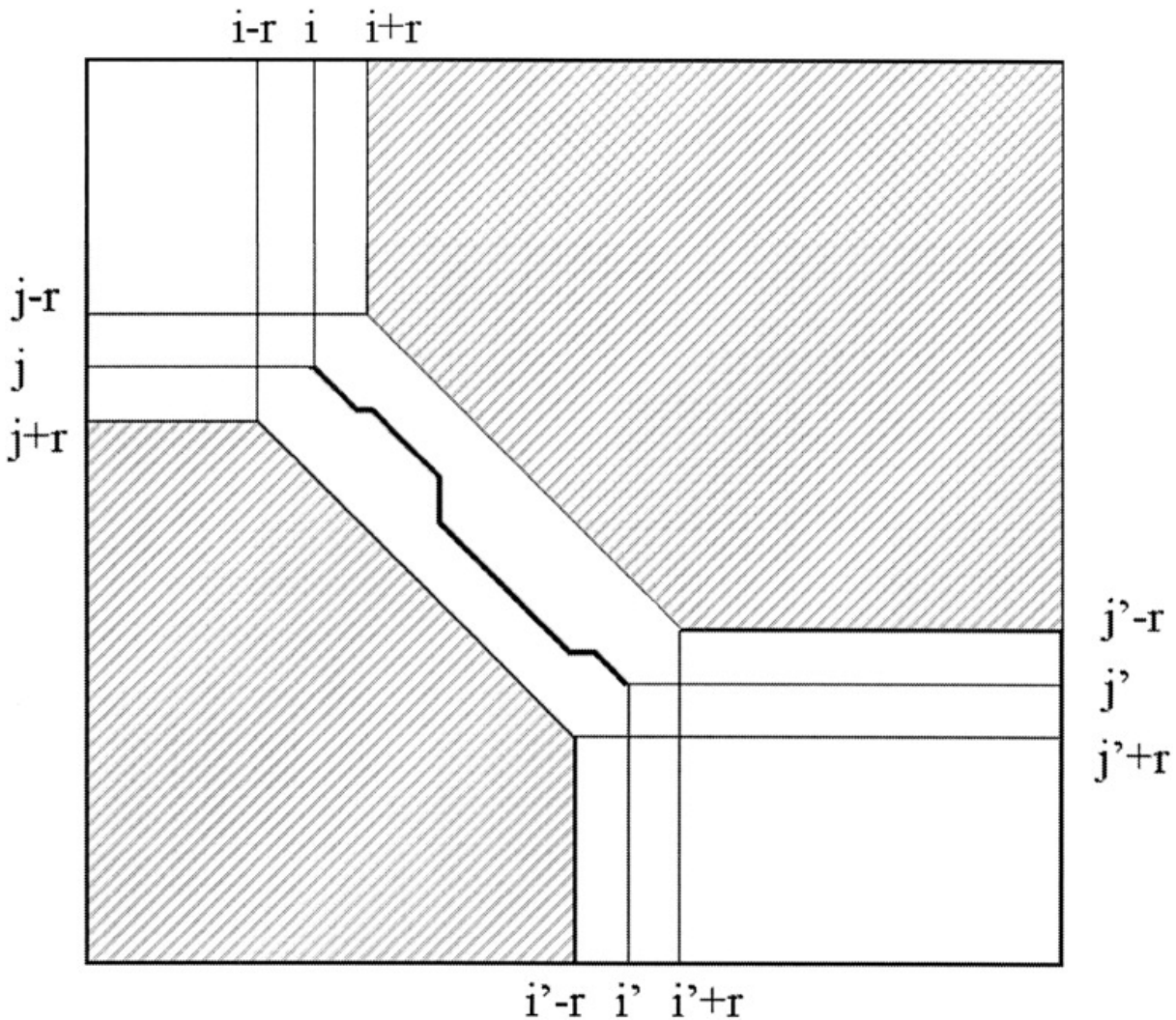


Figure from: Brudno et al. *Genome Research*, 2003

MUMmer overview

- Find anchors (Suffix tree - MUMs)
- Find good anchor chain (LIS variant)
- Align inter-anchor regions (Needleman-Wunsch, or if too big, recurse with smaller anchors)

AVID overview

- RepeatMask sequences
- Find anchors (suffix tree, exact & wobble)
- Find good chain of anchors (SW 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

LAGAN overview

- generate local alignments (CHAOS)
- find good chain of local alignments (sparse DP)
- recurse in between anchors in chain

- Finally, find full global alignment (limited-area DP)