Algorithms in Genome Analysis, Spring 2023

Veli Mäkinen

Week 6

Bidirectional BWT: maximal repeats, MUMs, overlaps revisited

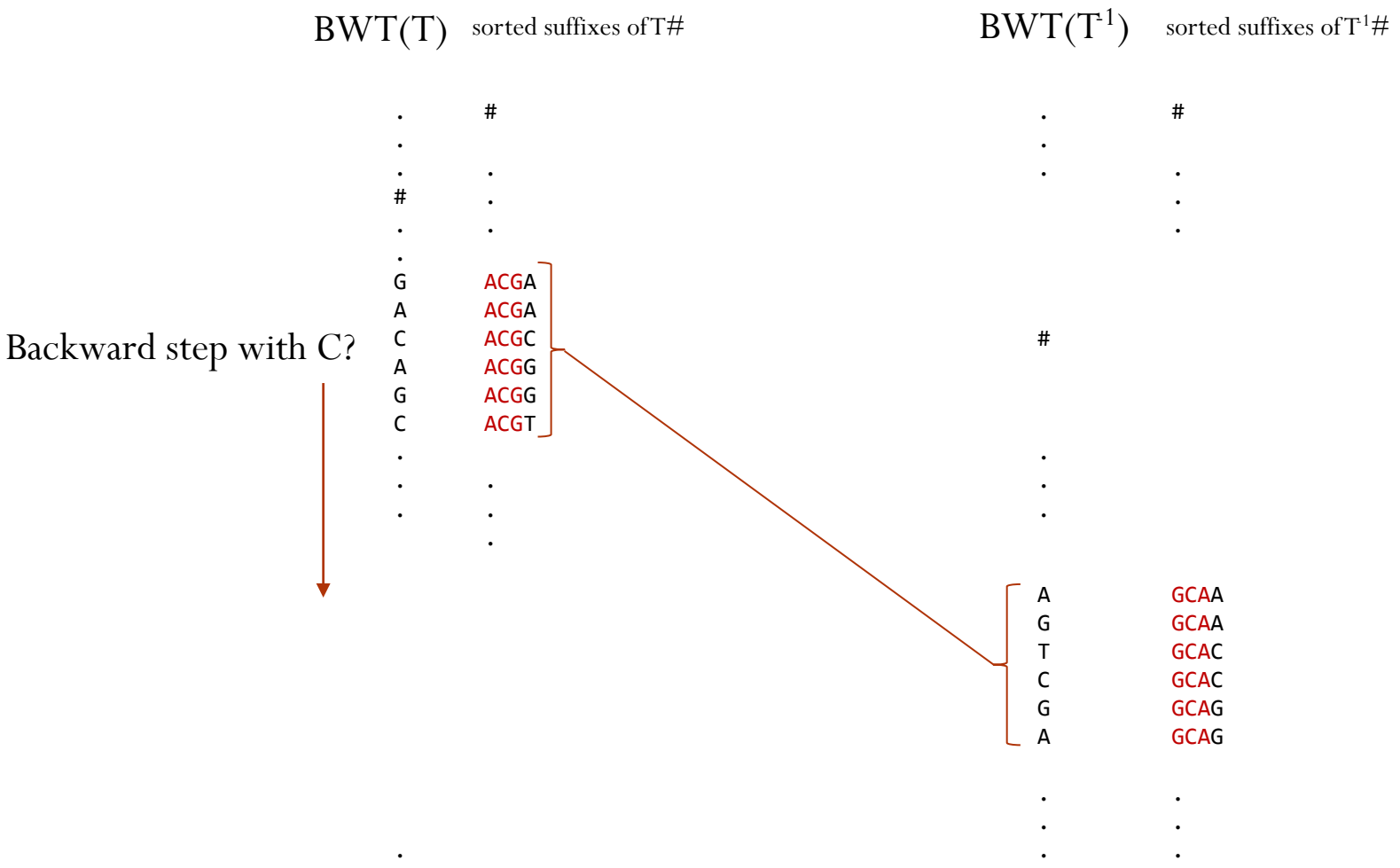
Bidirectional BWT

Let's motivate it first through case analysis pruning / read alignment

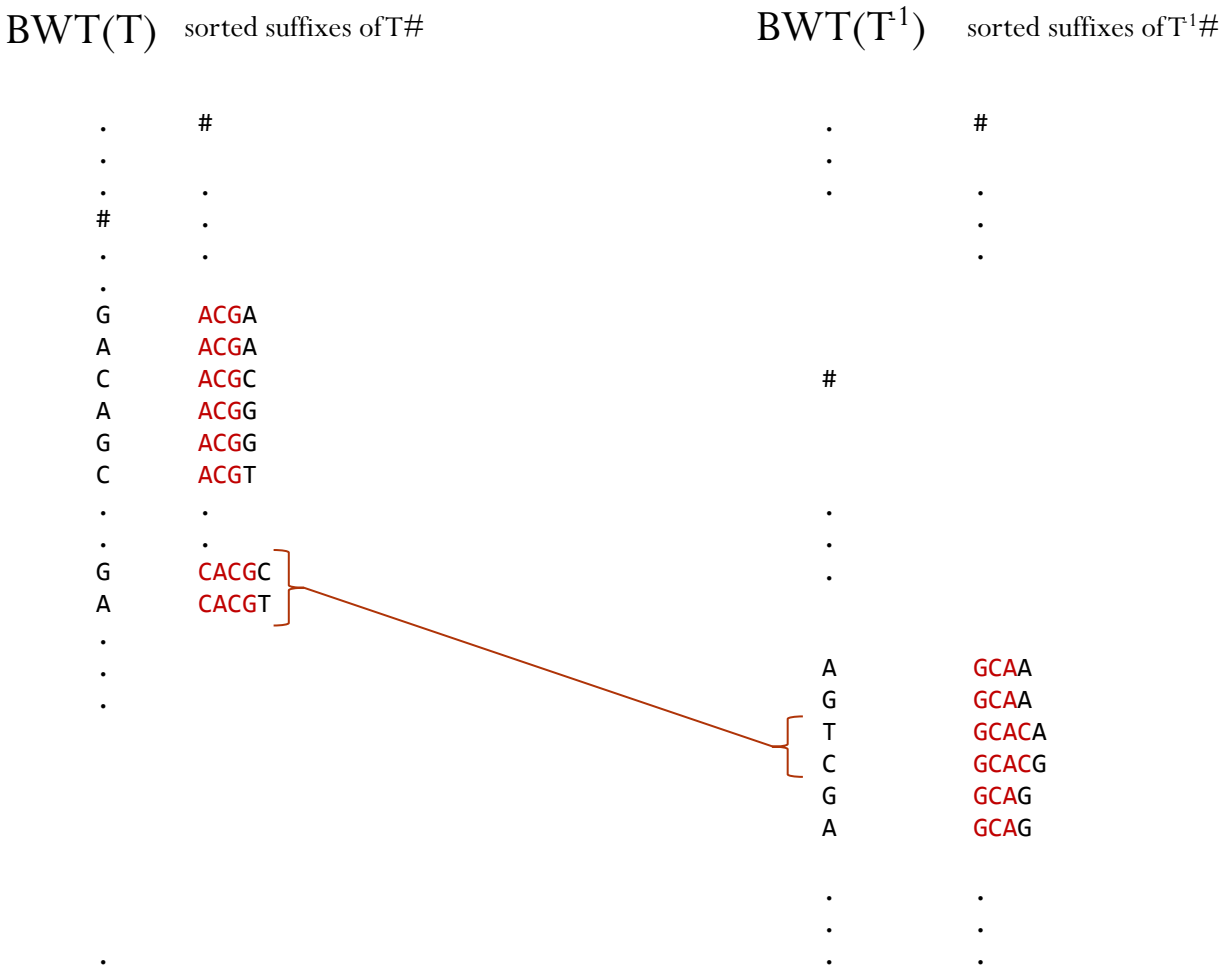
Case analysis pruning revisited

- Consider searching pattern P with 2 mismatches in text T
- For any partitioning of P to three pieces $P = P^1P^2P^3$ the mismatches between an occurrence T[i..j] will be distributed
  - 002,020,200,011,101,or 110, where the i-th digit denotes the number of mismatches in piece P^i.
  - Using indexes on BWT of T and on BWT of the reverse of T, it is possible to search P starting with allowing no mismatches in the first piece (no branching), except for the case 101.
  - For case 101 we need to start the search in the middle of P, e.g., searching P^1P^2 backwards (allowing one mismatch in part P^1), then continuing with P^3 forwards.
  - This change of direction is the key feature of the *bidirectional BWT index*.

Bidirectional BWT index



Bidirectional BWT index

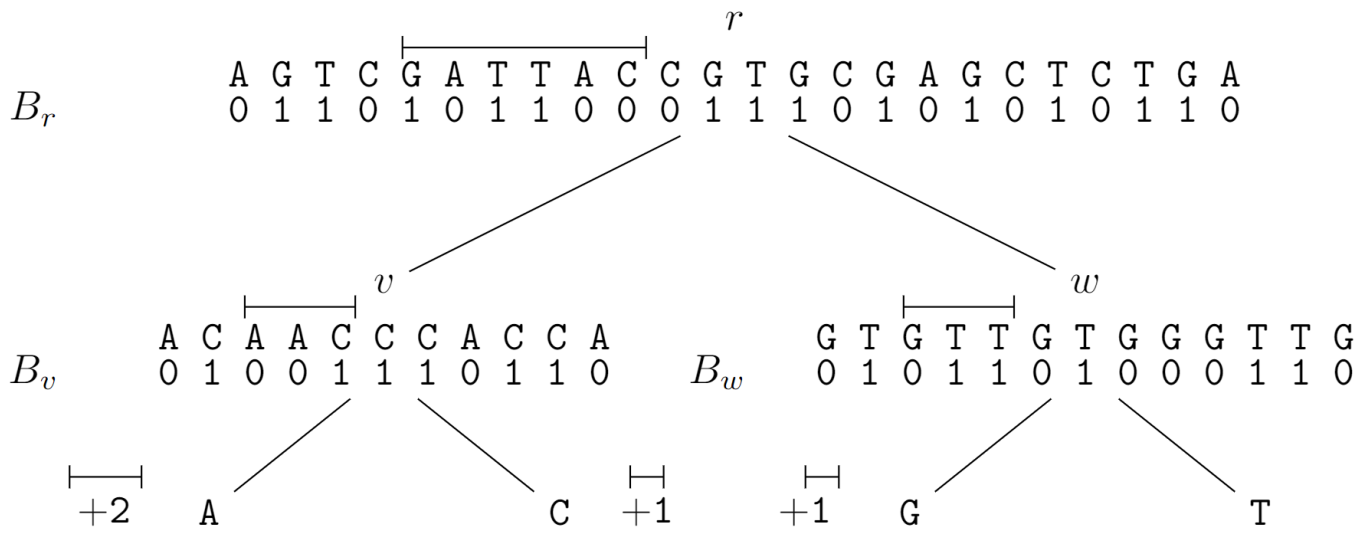


Range counting with wavelet tree

- Bidirectional backward step needs operation RangeCount(L,i,j,a,b) that gives the amount of symbols in range [a..b] occurring in L[i..j], where L is the Burrows-Wheeler transform of text T.
- Namely, let [i..j] and [x..y] be the BWT ranges corresponding to a strings Q and Q^{-1}, respectively, in BWT L of T and BWT L' of reverse of T.
- [i..j] can be updated to range [i'..j'] corresponding to cQ using the normal BWT backward step update rules.
- [x..y] can be updated to range [x'..y'] using
  - x' = x+RangeCount(L,i,j,0,c-1)
  - y' = x+RangeCount(L,i,j,0,c)-1
- Wavelet tree supports this operation in $O(\log|\Sigma|)$ time.

Example 3.3 Continue Example 3.2 to illustrate the computation of `rangeCount`$(T, 5, 10, 1, 3)$, that is, to count how many times characters `A`, `C`, and `G` occur in substring $T[5..10]$.

Solution



Here the ranges on top of the sequences illustrate $[i_v..j_v]$ for each node v. The ranges in the last level illustrate contributions to the range count. That is, `rangeCount`$(T, 5, 10, 1, 3) = 2 + 1 + 1 = 4$, as $L =$ `A`, $V' = \{$`C`$\}$, $V'' = \emptyset$, and $R =$ `G`.

Maximal repeats revisited

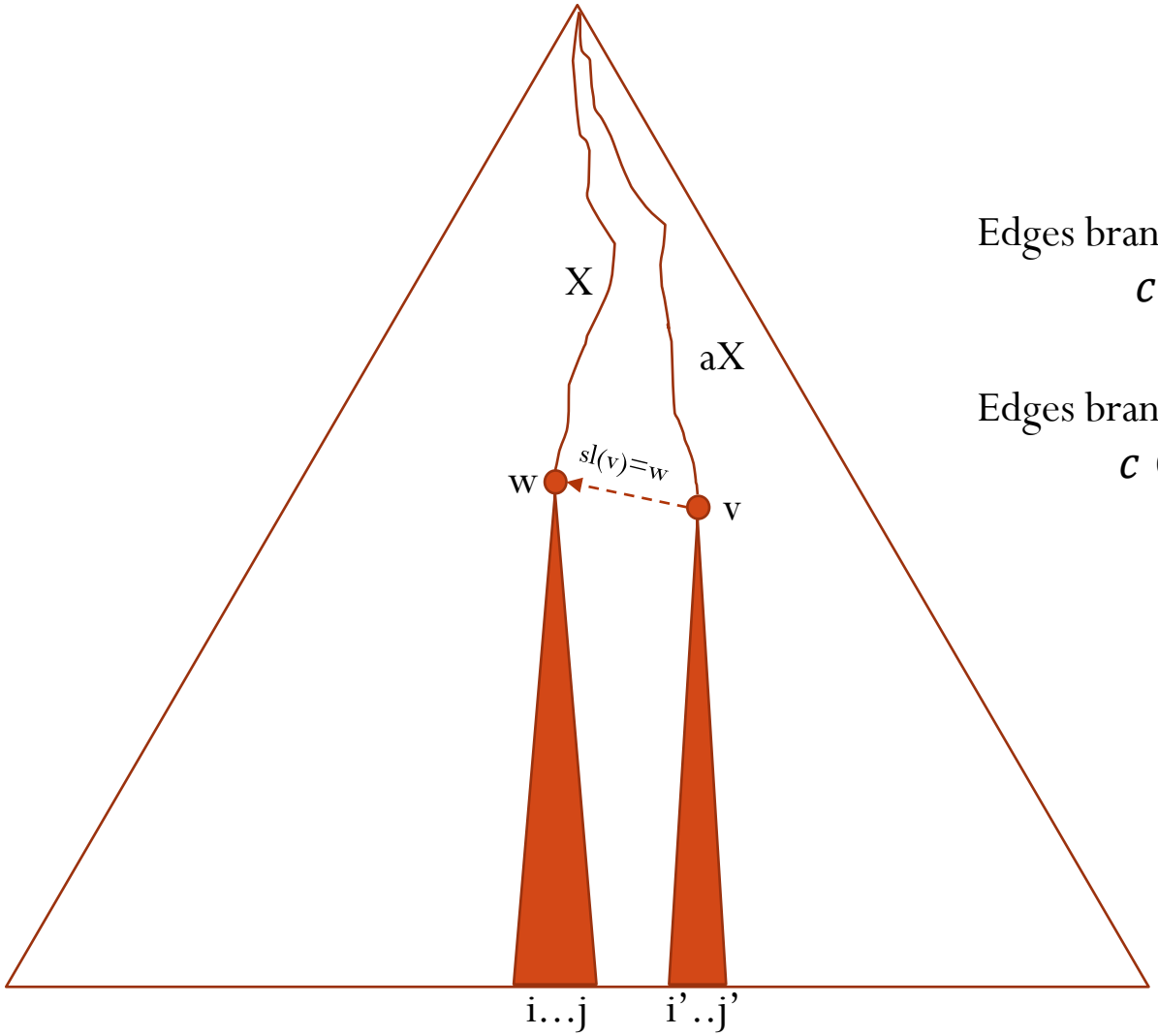
Bidirectional BWT yields space-efficient enumeration of maximal repeats

Visiting suffix tree nodes with bidirectional BWT 1/2

- Consider backward backtracking with bidirectional BWT of text T
  - Start with pair of ranges L[1..n], L'[1..n], where L is the BWT of T and L' is the BWT of the reverse of T
  - Do bidirectional backward step with all symbols
  - Continue backward steps recursively at each interval pair [i..j], [x..y] until yielding an empty interval pair or L'[x..y] contains only a run of single symbol
- We will see that this search implicitly visits all internal nodes of the suffix tree of T!
  - Hence, it works in $O(|T| \log \sigma)$ time and can turn many suffix tree algorithms space-efficient: From $O(|T|)$ words to $\sim 2|T| \log \sigma$ bits, that is, from hundreds of bits per nucleotide to ~5 bits per nucleotide for DNA

Visiting suffix tree nodes with bidirectional BWT 2/2

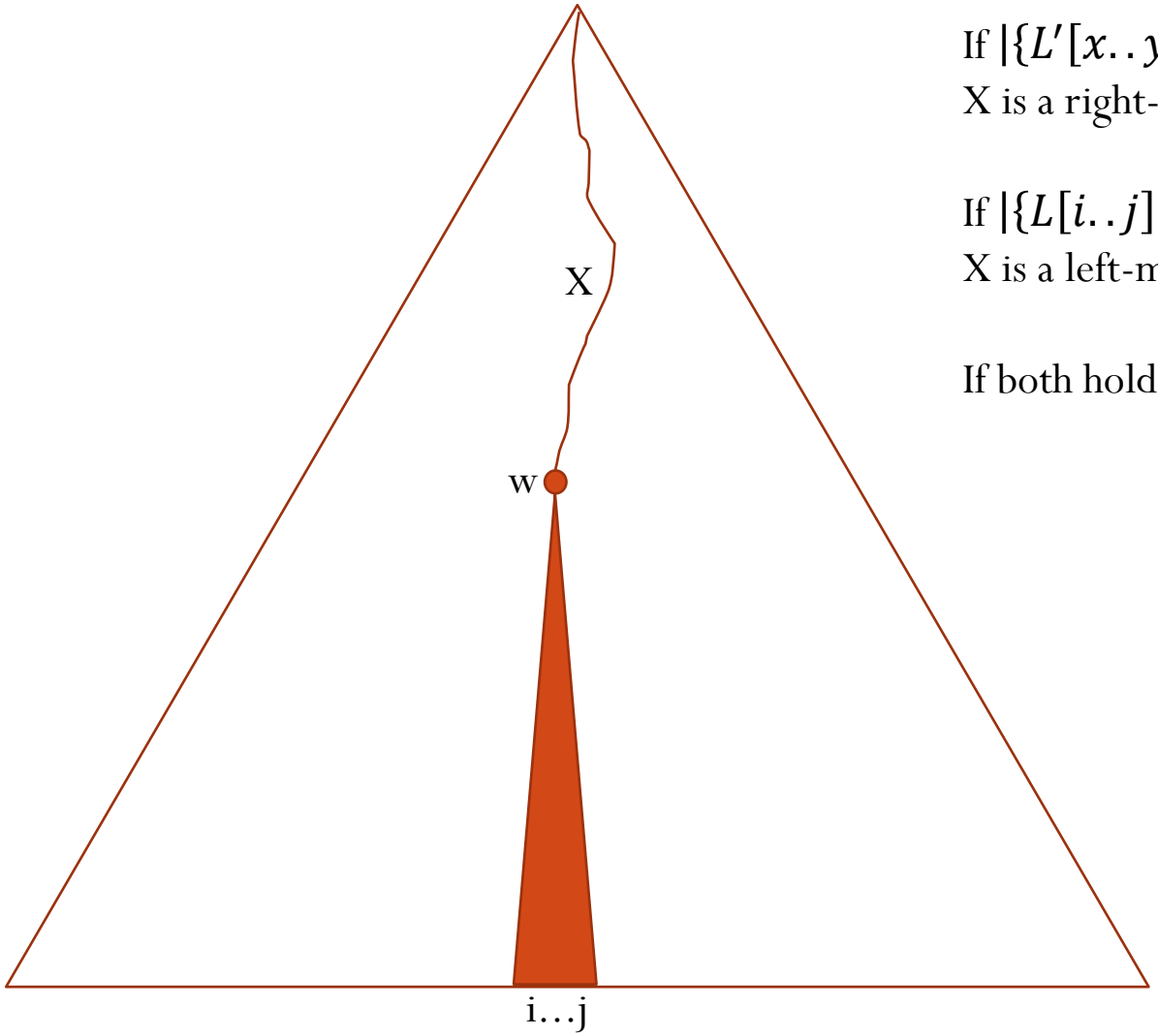
- Consider suffix tree of T#. Let aX and X be the strings spelled on the path from root to nodes v and w, respectively.
- Consider *suffix link* sl(v)=w.
- Assume inductively that our backtracking algorithm reaches interval pair [i..j], [x..y] such that subtree of w contains leaves with lexicographic ranks [i..j].
- Since w is an internal node, L'[x..y] is not a run of a single symbol, and the algorithm makes a backward step with symbol a reaching interval pair [i'..j'], [x'..y'].
- Suffixes with lexicographic ranks [i'..j'] are the only ones starting with aX, that is, those forming the leafs of subtree rooted at node v.
- Hence, the algorithm implicitly visits all suffix tree nodes.
- Last branch of recursion yields intervals that are not suffix tree nodes (L'[x..y] consists of a run of one symbol), but this does not affect the asymptotic running time (see course book)



Edges branching from w start with
$c \in \{L'[x..y]\}$

Edges branching from v start with
$c \in \{L'[x'..y']\}$

Maximal repeats



If $|\{L'[x..y]\}| > 1$ then
X is a right-maximal repeat

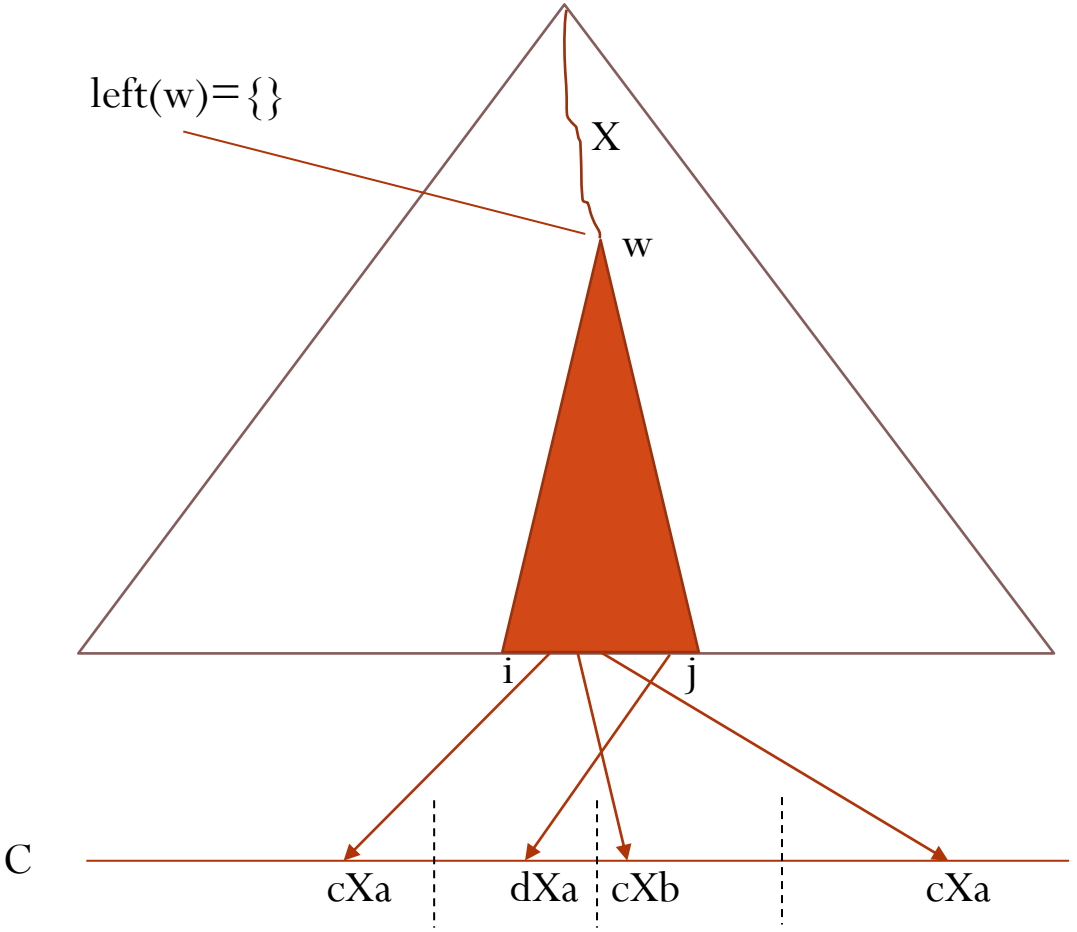
If $|\{L[i..j]\}| > 1$ then
X is a left-maximal repeat

If both hold, then X is maximal repeat

Maximal unique matches revisited

Bidirectional BWT yields space-efficient enumeration of maximal unique matches (MUMs)

MUMs



Consider bidirectional BWT index on the concatenation
$C = S^1\#S^2\#\cdots S^d\#$ of d sequences

Find max repeats X with interval pair [i..j], [x..y] s.t. j-i+1=d as candidates, omitting branches with #

Consider a bitvector B[1..n] initialized to zeros and bitvector I[1..n] s.t. I[i]=1 iff [i..j], [x..y] is the interval associated to some MUM candidate X

Consider reading C backwards using LF-mapping so that at each step we know the lexicographic rank k of some suffix C[p..], which starts at d'-th sequence:

- If k<select(rank(I,k))+d, set B[i+d'-1]=1

→

X is MUM iff B[i..i+d-1]=11..1

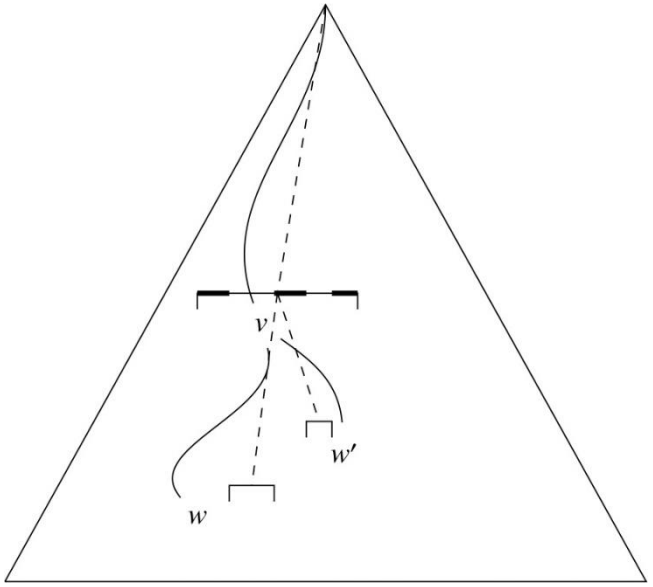
Maximal overlaps revisited

BWT yields space-efficient computation of maximal overlaps

Overlaps using BWT

- Consider BWT L on the concatenation $C = \#R^1\#R^2\#\cdots R^d\#$ of d reads
- Consider backward searching R^a. After at least κ steps, try to backward step with #. If this results into a non-empty range, some reads have long enough prefix that matches suffix of R^a
- Each occurrence of # can be associated with the id of the following read, so we can report the overlapping pairs
- The problem again is how to report only maximal overlaps when a read has multiple overlaps with another
- **Observation**: If read R^a has multiple overlaps with read R^b, the shorter overlaps are prefixes of the longer overlaps
- **Corollary**: The lexicographic ranges of the overlapping suffixes are nested

Overlaps using suffix tree and BWT



- Consider suffix tree of read R^a
- Consider having backward searched suffix R^a[p..] using BWT of C and backward step with # results into a non-empty interval [i..j]
- Store [i..j] at locus v of suffix tree, where the path to v spells R^a[p..]
- After reading whole R^a, collapse the suffix tree removing nodes not storing an interval
- Remove intervals of children from each parent interval
- What remains are the longest overlaps