Lecture 6

Introduction to bioinformatics (MVE510)

Autumn 2020

Additional reading: Mapping reads on a genomic sequence: an algorithmic overview and a practical comparative analysis. Schbath S, Martin V, Zytnicki M, Fayolle J, Loux V, and Gibrat JF. *Journal of Computational Biology*, 19(6) 2012.

Repetition

- The Needleman-Wunsch algorithm can be used to find global alignments. The best alignment is identified by iteratively fill a alignment matrix and backtrack from the highest value.
- The Smith-Waterman algorithm can be find local alignments. The best alignment is found similarly to the NW algorithm but where all negative values has been replaced by zeros and the alignment can start and end at any positions.

Repetition

- BLAST uses a ‘seed-and-extend’ algorithm to efficiently calculate local alignments. This makes it 50 times fast than Smith-Waterman.
- BLAST is however still to slow to efficiently map reads from next generation sequencing to a reference.

The BLAST algorithm (Basic Local Alignment Search Tool)

Is BLAST fast enough?

No! Next generation DNA sequencing produce too many reads (50x faster than SW is not enough!).

Can we further improve the speed?

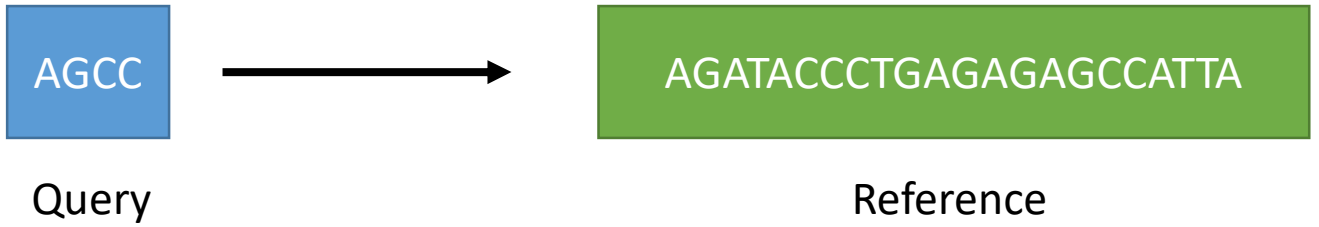
Yes, but we need some additional concepts from computer science!

What is an index?

- An index is a data structure that enable fast lookup of keys
- In our case the data structure will built form the reference sequence and key our query sequence
- Many form of indices assume that the reference is long (e.g. a genome) and the query sequence is short (e.g. a NGS read)
- Once the index is built, we can use it to lookup to find where specific read match the reference.
- We will focus on exact matches but it is also possible to include mismatches and gaps.

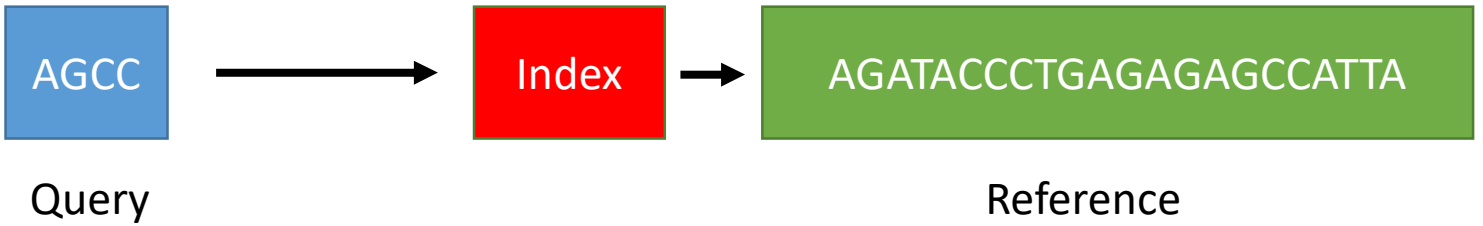
Index-based read search

Naive search



Slow: $O(L_R L_G)$

Index-based search



Fast!

Suffix trees

- Are highly efficient to find exact matches in a sequence. The complexity is

$$O(L_R)$$

  for each read.

- Note that L_G is gone (!!!). The size of the reference does no longer matter!

Suffix trees

BUT…. (there is no free lunch!)

- Building the suffix trees takes time ($O(L_G)$) but we only need to do this once.
- Suffix trees are very large. The suffix tree for the entire human genome is so large that it can not be stored in the memory of a standard computers.
- If we want to include mismatches and gaps, the complexity increases fast!

Example

Reference: GAGAGGCAGC$

Suffixes	Position
GAGAGGCAGC$	1
AGAGGCAGC$	2
GAGGCAGC$	3
AGGCAGC$	4
GGCAGC$	5
GCAGC$	6
CAGC$	7
AGC$	8
GC$	9
C$	10
$	11

Example

Reference: GAGAGGCAGC$

Sorted suffixes	Sorted position
AGAGGCAGC$	2
AGC$	8
AGGCAGC$	4
CAGC$	7
C$	10
GAGAGGCAGC$	1
GAGGCAGC$	3
GCAGC$	6
GC$	9
GGCAGC$	5
$	11

Example

Reference: GAGAGGCAGC$

Sorted suffixes	Sorted position	Cylinder suffix array	BW-transform
AGAGGCAGC$	2	AGAGGCAGC$G	G
AGC$	8	AGC$GAGAGGC	C
AGGCAGC$	4	AGGCAGC$GAG	G
CAGC$	7	CAGC$GAGAGG	G
C$	10	C$GAGAGGCAG	G
GAGAGGCAGC$	1	GAGAGGCAGC$	$
GAGGCAGC$	3	GAGGCAGC$GA	A
GCAGC$	6	GCAGC$GAGAG	G
GC$	9	GC$GAGAGGCA	A
GGCAGC$	5	GGCAGC$GAGA	A
$	11	$GAGAGGCAGC	C

1	2	3	4	5	6
$	A	$A	AC	$AC	ACG
G	A	GA	AC	GAC	ACT
A	C	AC	CG	ACG	CGA
A	C	AC	CT	ACT	CT$
C	G	CG	GA	CGA	GAC
C	T	CT	T$	CT$	T$A
T	$	T$	$A	T$A	$AC

7	8	9	10	11
$ACG	ACGA	$ACGA	ACGAC	$ACGAC
GACT	ACT$	GACT$	ACT$A	GACT$A
ACGA	CGAC	ACGAC	CGACT	ACGACT
ACT$	CT$A	ACT$A	CT$AC	ACT$AC
CGAC	GACT	CGACT	GACT$	CGACT$
CT$A	T$AC	CT$AC	T$ACG	CT$ACG
T$AC	$ACG	T$ACG	$ACGA	T$ACGA

12

ACGACT
ACT$AC
CGACT$
CT$ACG
GACT$A
T$ACGA
$ACGAC

13

$ACGACT
GACT$AC
ACGACT$
ACT$ACG
CGACT$A
CT$ACGA
T$ACGAC

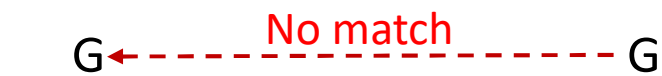
14

ACGACT$
ACT$ACG
CGACT$A
CT$ACGA
GACT$AC
T$ACGAC
$ACGACT

Example

Use the BW-transform to find GCAG in GAGAGGCAGC$.

	BW-transform	Sorted BW-transform	Sorted position
	G	A	2
	C	A	8
	G	A	4
	G	C	7
	G	C	10
	$	G	1
1	A	G	3
	G	G	6
2	A	G	9
3	A	G	5
	C	$	11



Example

Use the BW-transform to find GCAG in GAGAGGCAGC$.

	BW-transform		Sorted BW-transform		Sorted position
	G	No match	A	1	2
1	C		A	2	8
	G	No match	A	3	4
	G		C		7
	G		C		10
	$		G		1
	A		G		3
	G		G		6
	A		G		9
	A		G		5
	C		$		11

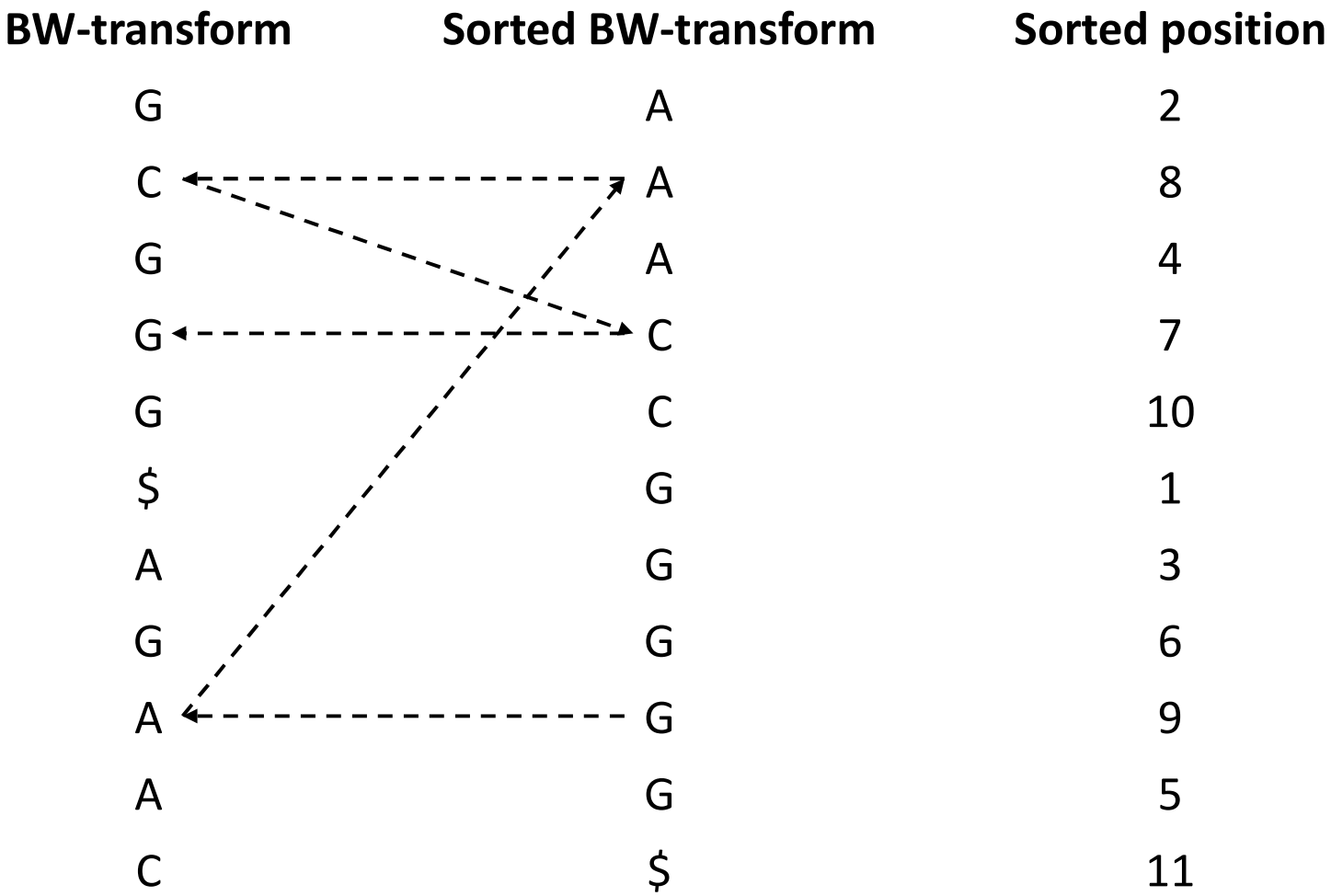
Example

Use the BW-transform to find GCAG in GAGAGGCAGC$.

BW-transform	Sorted BW-transform		Sorted position
G	A		2
C	A		8
G	A		4
G ←	C	1	7
G	C		10
$	G		1
A	G		3
G	G		6
A	G		9
A	G		5
C	$		11

Example

Use the BW-transform to find GCAG in GAGAGGCAGC$.



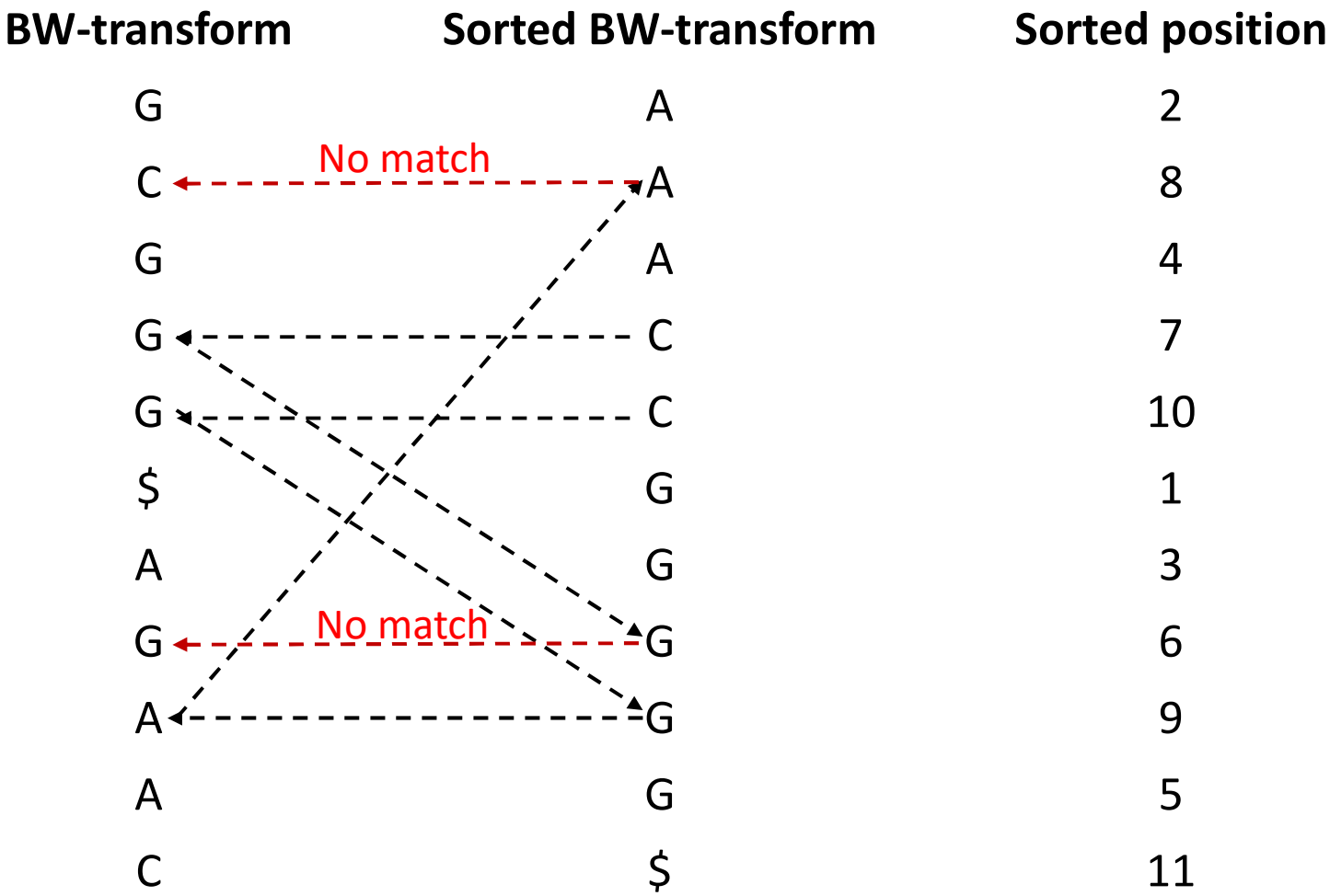
Example

Use the BW-transform to find GAGC in GAGAGGCAGC$.

BW-transform	Sorted BW-transform	Sorted position
G	A	2
C	A	8
G	A	4
G	C	7
G	C	10
$	G	1
A	G	3
G	G	6
A	G	9
A	G	5
C	$	11

Example

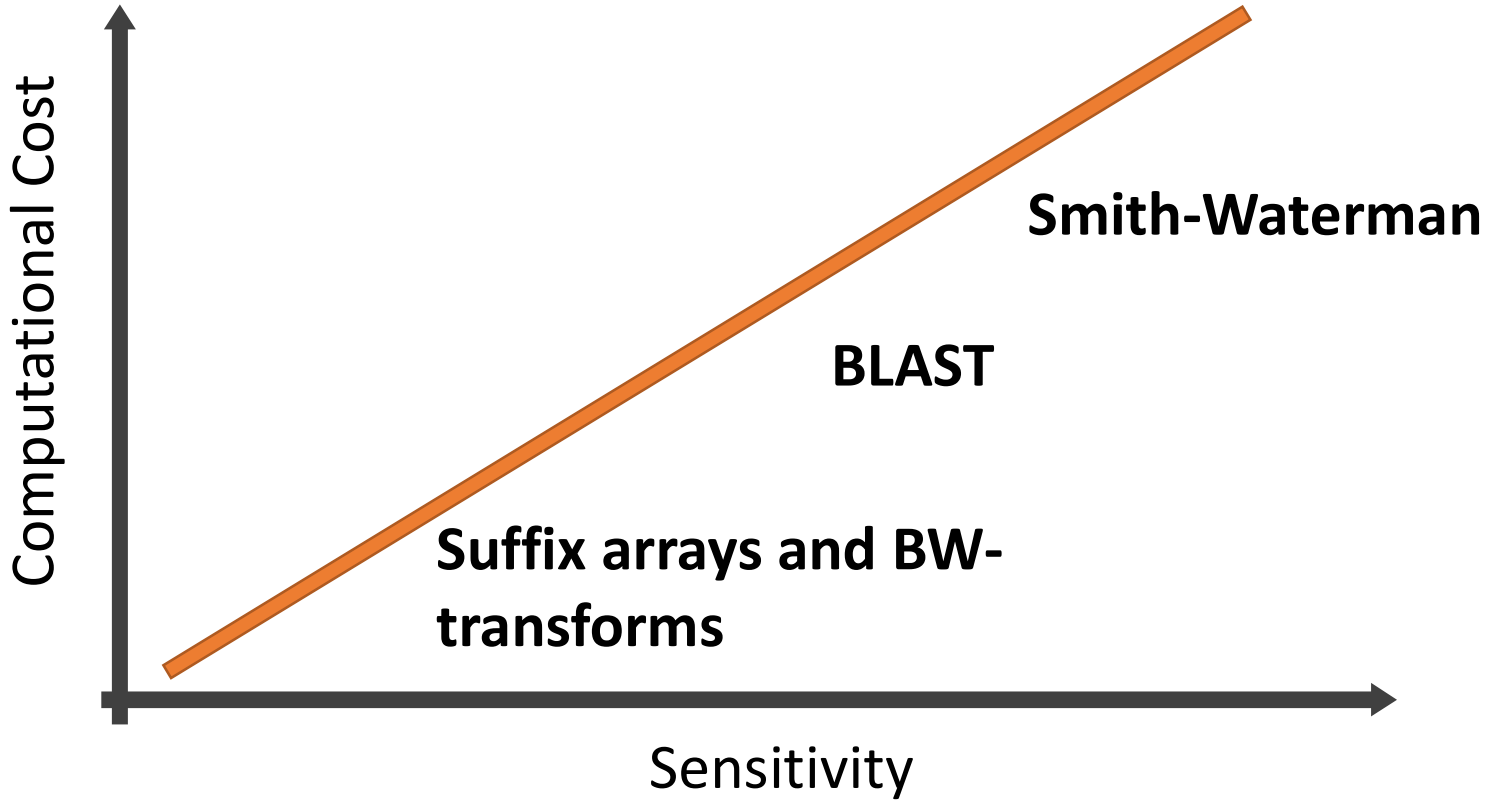
Use the BW-transform to find GAGC in GAGAGGCAGC$.



The Borrows-Wheeler Aligner (BWA)

- Based on seed-and-extend approach.
- Seeds are identified using suffix arrays and the Burrows-Wheeler transform.
- Seeds are extended into full alignments using more accurate algorithms.
- Optimized for large volumes of reads from next generation DNA sequencing.
- You will use this aligner in the computer exercises.

Choosing a suitable aligner



Choosing a suitable aligner

Smith-Waterman

Slow but highly sensitive.

Software: FASTA and EMBOSS.

Seed and extend

Faster but have reduced sensitivity.

Software: BLAST

Choosing a suitable aligner

Suffix-array and Burrow-Wheeler transform

Very fast, low sensitivity. But often suitable for mapping sequence read to a reference.

Software: BWA

Summary of this lecture

- An index is a data structure that enables fast lookup of exact matches in a DNA sequence.
- Suffix trees are crated from the suffixes of a DNA sequence. By traversing the tree we can find the position of any subsequence.
- The Burrows-Wheeler transform orders the genome in a specific way that makes identification of subsequences highly efficient.

Summary of this lecture

- Read alignment using suffix arrays and the Burrows-Wheeler transform are as fast as a suffix tree but does not require the entire tree to be created and stored in memory.