Sequence Analysis and Genomics

6 · Sequencing data: formats, quality control, read mapping

Maria Waldl Elisa Israel

Bioinformatics, Leipzig University
elisa.israel@uni-leipzig.de

maria.waldl@uni-leipzig.de

16/11/2026 · WiSe 2026/27 · module 10-202-2207

Today

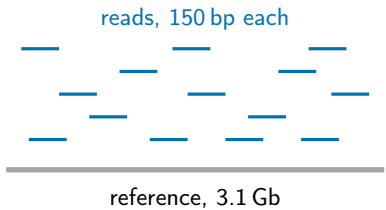
After today you can ...

- convert between Phred scores and error probabilities
- read a FastQC report and infer what went wrong upstream
- explain why read mapping needs an index, and compare k-mer hash, suffix array and FM-index by query time and memory
- explain how backward search finds a pattern with the FM-index, and why it needs so little memory
- explain seed-and-extend as a combination of seeding (BLAST) and banded DP
- interpret MAPQ, and explain why reads from repeats get low MAPQ
- read a SAM record: FLAG, paired-end information, CIGAR
- name what has to be recorded so that an analysis can be rerun

One sequencing run

- One run of a large short-read instrument: **billions of reads**, 150 bp each, often in pairs
- One human genome at 30× coverage: $\frac{30 \times 3.1 \cdot 10^9\,\text{bp}}{150\,\text{bp}} \approx 6 \cdot 10^8$ reads
- The reference: 3.1 Gb

For every read: *where in the genome does it come from?*



Smith-Waterman for every read?

Your turn 3 min

How long would it take to align 10^9 reads of 150 bp to the human genome with Smith-Waterman (lecture 1)? An optimised implementation fills about 10^{10} cells per second.

Smith-Waterman for every read?

Your turn 3 min

How long would it take to align 10^9 reads of 150 bp to the human genome with Smith-Waterman (lecture 1)? An optimised implementation fills about 10^{10} cells per second.

per read: $150 \times 3 \cdot 10^9 \approx 5 \cdot 10^{11}$ cells

all reads: $10^9 \times 5 \cdot 10^{11} = 5 \cdot 10^{20}$ cells

time: $5 \cdot 10^{20} / 10^{10}\,\mathrm{s}^{-1} = 5 \cdot 10^{10}\,\mathrm{s} \approx$ **1 600 years**

Smith-Waterman for every read?

Your turn 3 min

How long would it take to align 10^9 reads of 150 bp to the human genome with Smith-Waterman (lecture 1)? An optimised implementation fills about 10^{10} cells per second.

$$\begin{aligned}
\text{per read:}\quad & 150 \times 3 \cdot 10^9 \approx 5 \cdot 10^{11} \text{ cells} \\
\text{all reads:}\quad & 10^9 \times 5 \cdot 10^{11} = 5 \cdot 10^{20} \text{ cells} \\
\text{time:}\quad & 5 \cdot 10^{20} / 10^{10}\,\text{s}^{-1} = 5 \cdot 10^{10}\,\text{s} \approx \mathbf{1\,600\ years}
\end{aligned}$$

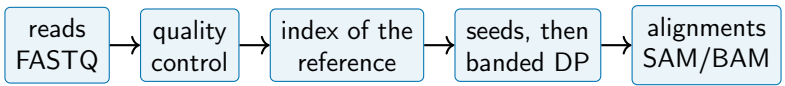
The DP is correct and unusable at this scale: we need a **data structure**, not a faster computer.

BLAST already cheated

Lecture 2: BLAST does not align the query against the whole database.

1. find short **seeds** (words) shared by query and database
2. **extend** only around the seeds

Today the same plan, made exact and fast:



Short and long reads

	read length	**errors**	**error type**
Illumina	100–300 bp	few (well below 1 %), more towards the 3' end	substitutions
Nanopore	10–100 kb	a few %	indels (homopolymers)
PacBio HiFi	15–20 kb	below 0.1 %	few, indels

- QC: quality along the read matters for short reads (next frames)
- Mapping: substitutions are cheap for a seed-based mapper; frequent indels need a different seeding (long-read mappers)

A FASTQ record

```
@SRR2420871.3 3/1                                          header: read name, mate (/1)
GCGCTGGAAACGTGGCTGCTGCCA ... TGGAGTTGCNNTCACCTGCCCAAA      sequence, 301 bases
+                                                          separator
CCCCCGGGGGGGGFEGGGGGGGGG ... 9=FFBF56)##--((-8>028::0      quality per base
```

First and last 24 of 301 bases; E. coli, MiSeq, the exercise QC data. `N`: no base called, quality # ($Q = 2$).

- Paired-end: two files, `_1` and `_2`, **the same order**: record k in one file is the mate of record k in the other
- Usually compressed: `reads_1.fq.gz`

⚠ Careful

`@` is also a quality character ($Q = 31$). Never parse FASTQ by looking for lines that start with `@`: count four lines per record.

Phred scores

Definition: Phred quality

$$Q = -10 \log_{10} P_{\text{error}} \qquad \Longleftrightarrow \qquad P_{\text{error}} = 10^{-Q/10}$$

P_{error}: the base caller's estimate that this base is wrong.

Q	P_{error}	one error in	character
10	0.1	10 bases	+
20	0.01	100 bases	5
30	0.001	1 000 bases	?
40	0.0001	10 000 bases	I

Every 10 units of Q: ten times fewer errors. Remember this scale: it comes back later today for mapping quality.

Qualities as characters

One character per base: **ASCII code − 33 = Q**

character	#	+	5	?	@	I
ASCII	35	43	53	63	64	73
Q	2	10	20	30	31	40

- 33 = the first printable character (`!`, $Q = 0$)
- Old Illumina files (before 2011) used +64

⚠ Careful: wrong offset

Nothing crashes: all qualities are shifted by 31. FastQC guesses the encoding from the range of characters it sees.

Check: what does this read promise?

The last six bases of a read: `II?5+#`

character	I	I	?	5	+	#
ASCII	73	73	63	53	43	35

Your turn 3 min

Decode the qualities. How many errors do you expect in these 6 bases?

Check: what does this read promise?

The last six bases of a read: `II?5+#`

character	I	I	?	5	+	#
ASCII	73	73	63	53	43	35
Q	40	40	30	20	10	2

Your turn 3 min

Decode the qualities. How many errors do you expect in these 6 bases?

Check: what does this read promise?

The last six bases of a read: `II?5+#`

character	I	I	?	5	+	#
ASCII	73	73	63	53	43	35
Q	40	40	30	20	10	2
P_{error}	0.0001	0.0001	0.001	0.01	0.1	0.631

Your turn 3 min

Decode the qualities. How many errors do you expect in these 6 bases?

Check: what does this read promise?

The last six bases of a read: `II?5+#`

character	I	I	?	5	+	#
ASCII	73	73	63	53	43	35
Q	40	40	30	20	10	2
P_{error}	0.0001	0.0001	0.001	0.01	0.1	0.631

Your turn 3 min

Decode the qualities. How many errors do you expect in these 6 bases?

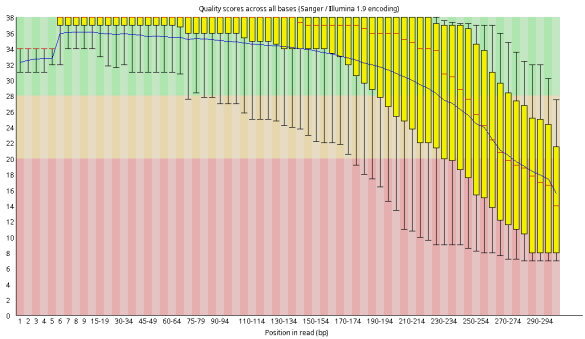
Expected errors $= \sum_i P_i = 0.74$. From the mean, $\bar{Q} = 23.7$, you would get $6 \times 10^{-\bar{Q}/10} = 0.026$: **the mean of Q is not the mean of P.**

From sample to FASTQ: causes and symptoms

cause	→ **symptom in the report**
molecules in a cluster get out of step (phasing)	→ quality drops along the read
fragment (insert) shorter than the read	→ adapter sequence at the 3' end
contamination (other organism, primers)	→ second peak in the GC content
many PCR cycles, little input DNA	→ many identical reads (duplicates)

- FastQC shows symptoms; the diagnosis needs the cause
- Not every symptom is a fault: the first two rows happen in every run
- Reading a full report: exercise 6

Quality along the read



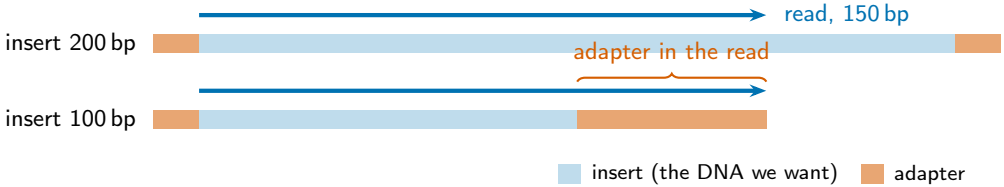
FastQC, read 2 of the exercise QC data (MiSeq, 301 cycles)

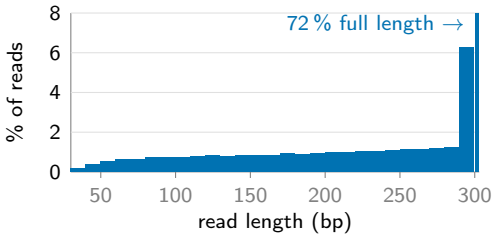
Why quality drops: a cluster is $\sim$1000 copies of one fragment, read in lockstep.

- in every cycle a few copies fall behind or run ahead
- the signal becomes a mixture of neighbouring bases
- the base caller is less sure: Q drops

Reading the plot: box per position; the median matters, and how low the lower whisker goes.

Adapters





Read 1, exercise QC data: 28 % cut at the adapter on the instrument

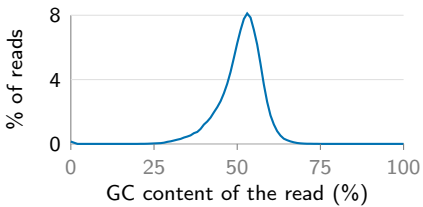
The read does not stop where the insert ends: the last 50 bases are adapter, not genome. They will not align, or align wrongly.

Expected, not an error: every library has a spread of insert sizes, and the short ones read into the adapter. Trim it.

GC content and duplicates

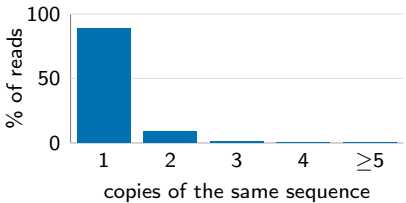
GC content per read

- one genome: one peak
- second peak: second source of DNA



Duplication level

- genome (WGS): random starts, few identical reads
- many: PCR copies of few molecules



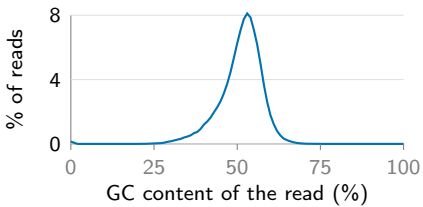
Your turn 2 min

An RNA-seq run shows 60 % duplicates. Is the library broken?

GC content and duplicates

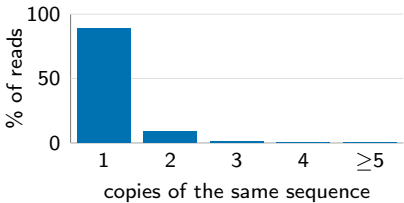
GC content per read

- one genome: one peak
- second peak: second source of DNA



Duplication level

- genome (WGS): random starts, few identical reads
- many: PCR copies of few molecules



Your turn 2 min

An RNA-seq run shows 60 % duplicates. Is the library broken?

Not necessarily: a highly expressed gene gives many identical reads without any PCR problem. **Same plot, different cause.**

Trimming: what it costs

Trimmers (here: fastp) cut adapters and low-quality ends, and drop reads that get too short.

gain

- fewer wrong bases
- no adapter sequence in the alignment

cost

- shorter reads: harder to place uniquely
- lost reads, and pairs that lose a mate

Your turn 3 min

Should we trim every read until all bases have $Q \geq 30$?

After trimming: **run FastQC again** and compare. Did the trimmer do what you wanted?

Trimming: what it costs

Trimmers (here: fastp) cut adapters and low-quality ends, and drop reads that get too short.

gain

- fewer wrong bases
- no adapter sequence in the alignment

cost

- shorter reads: harder to place uniquely
- lost reads, and pairs that lose a mate

Your turn 3 min

Should we trim every read until all bases have $Q \geq 30$?

Depends on the next step. Mappers align locally and soft-clip bad ends anyway (CIGAR, later today); an assembler or a variant caller may need cleaner reads. **There is no universal threshold**: whichever you pick, report it.

After trimming: **run FastQC again** and compare. Did the trimmer do what you wanted?

One question, five answers

Where does pattern P (length m) occur in text T (length n)?

- T: the reference, $n = 3 \cdot 10^9$, built **once**
- P: a seed from a read, $m \approx 20$, asked **billions of times**
- Naive: compare P at every position of T, $O(nm)$ per query

Today, by shrinking memory: k-mer table $\rightarrow$ suffix tree $\rightarrow$ suffix array $\rightarrow$ FM-index

All examples on one toy text: $T =$ `GATTACATTA$` ($: end marker, smaller than every letter)

A k-mer index

$T =$ `GATTACATTA$`, $k = 3$:

3-mer	positions
`ACA`	4
`ATT`	1, 6
`CAT`	5
`GAT`	0
`TAC`	3
`TTA`	2, 7

- every k-mer of $T \rightarrow$ list of its start positions
- query: one lookup, e.g. `TTA` $\rightarrow$ 2, 7
- BLAST builds this table for the query; mappers build it for the reference

What k costs

- Memory: one stored position per position of T:
  $3 \cdot 10^9 \times 4$ bytes $= 12$ GB, plus a table of 4^k buckets
- Only patterns of length exactly k

Your turn 3 min

1. Why not $k = 150$, the whole read?
2. Why not $k = 8$?

What k costs

- Memory: one stored position per position of T:
  $3 \cdot 10^9 \times 4$ bytes $= 12$ GB, plus a table of 4^k buckets
- Only patterns of length exactly k

Your turn 3 min

1. Why not $k = 150$, the whole read?
2. Why not $k = 8$?

1. one sequencing error or SNP and the read is not found; and the table needs one entry for each of $3 \cdot 10^9$ different 150-mers
2. $4^8 = 65\,536$ buckets for $3 \cdot 10^9$ positions: $\sim$45 000 hits per lookup, mostly by chance

Suffixes

suffixes by start		sorted	
0	GATTACATTA$	$	10
1	ATTACATTA$	A$	9
2	TTACATTA$	ACATTA$	4
3	TACATTA$	ATTA$	6
4	ACATTA$	ATTACATTA$	1
5	CATTA$	CATTA$	5
6	ATTA$	GATTACATTA$	0
7	TTA$	TA$	8
8	TA$	TACATTA$	3
9	A$	TTA$	7
10	$	TTACATTA$	2

- Every occurrence of P is the **start of a suffix**

Suffixes

suffixes by start		sorted	
0	GATTACATTA$	$	10
1	ATTACATTA$	A$	9
2	TTACATTA$	ACATTA$	4
3	TACATTA$	ATTA$	6
4	ACATTA$	ATTACATTA$	1
5	CATTA$	CATTA$	5
6	ATTA$	GATTACATTA$	0
7	TTA$	TA$	8
8	TA$	TACATTA$	3
9	A$	TTA$	7
10	$	TTACATTA$	2

- Every occurrence of P is the **start of a suffix**
- In sorted order, the suffixes that start with TTA are **adjacent**: one interval

The suffix array

row	SA	suffix
0	10	`$`
1	9	`A$`
2	4	`ACATTA$`
3	6	`ATTA$`
4	1	`ATTACATTA$`
5	5	`CATTA$`
6	0	`GATTACATTA$`
7	8	`TA$`
8	3	`TACATTA$`
9	7	`TTA$`
10	2	`TTACATTA$`

Definition: Suffix array

$SA[r]$ = start of the suffix in row r: $SA = [10, 9, 4, 6, 1, 5, 0, 8, 3, 7, 2]$

- store SA and T: 4 bytes per base + text
- query: binary search, compare P with the suffix in the middle row
- $O(m \log n)$ per query

Your turn 3 min

Find the interval of `TTA` by binary search: start at row 5.

The Burrows-Wheeler transform

row	SA	rotation	L
0	10	$GATTACATTA	A
1	9	A$GATTACATT	T
2	4	ACATTA$GATT	T
3	6	ATTA$GATTAC	C
4	1	ATTACATTA$G	G
5	5	CATTA$GATTA	A
6	0	GATTACATTA$	$
7	8	TA$GATTACAT	T
8	3	TACATTA$GAT	T
9	7	TTA$GATTACA	A
10	2	TTACATTA$GA	A

Wrap each suffix around the end of T: every row becomes a **rotation** of T (grey: the wrapped part).

Definition: BWT

L = **last column** of the sorted rotations, $L[r] = T[SA[r] - 1]$: the character **before** each suffix

$BWT(T) = L =$ ATTCGA$TTAA

- nothing new is sorted: rotations sort like suffixes ($ is unique)
- same characters as T, different order

F and L

Two columns of the sorted rotations:

- F: the **first** column = all characters of T, sorted: `$AAAACGTTTT`
- L: the **last** column, the BWT: `ATTCGA$TTAA`
- in every row, L comes right before F in T (the rotation wraps around)

F needs no storage

$C[c]$ = number of characters in T smaller than c:
$C[\texttt{A}] = 1, \ldots, C[\texttt{T}] = 7$;
rows $C[c]$ onwards start with c

L compresses well

a repeated word (`TTA`, `ATTA`) puts its suffixes next to each other, and they are preceded by the same character: runs in L

LF-mapping

	F	L
0	$ GATTACATT	A
1	A $GATTACAT	T
2	A CATTA$GAT	T
3	A TTA$GATTA	C
4	A TTACATTA$	G
5	C ATTA$GATT	A
6	G ATTACATTA	$
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A
10	T TACATTA$G	A

LF-mapping

	F	L
0	$ GATTACATT	A_1
1	A_1 $GATTACAT	T
2	A_2 CATTA$GAT	T
3	A_3 TTA$GATTA	C
4	A_4 TTACATTA$	G
5	C ATTA$GATT	A_2
6	G ATTACATTA	$
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A_3
10	T TACATTA$G	A_4

Definition: LF-mapping

The k-th A in L is the k-th A in F: the same A of T.

LF-mapping

	F	L
0	$ GATTACATT	A_1
1	A_1 $GATTACAT	T
2	A_2 CATTA$GAT	T
3	A_3 TTA$GATTA	C
4	A_4 TTACATTA$	G
5	C ATTA$GATT	A_2
6	G ATTACATTA $	
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A_3
10	T TACATTA$G	A_4

Definition: LF-mapping

The k-th A in L is the k-th A in F: the same A of T.

Why:

- the rows ending in A (rows 0, 5, 9, 10) are sorted by what comes *before* that A: the rest of their rotation
- rotate each of them by one: the A moves to the front, the rest stays the same
- the rows starting with A are sorted by that same rest: **same order**

Backward search: the idea

Goal: the interval of rows whose suffixes start with P, **without** looking at the suffixes.

1. Read P from **right to left**
2. Keep the interval of rows that start with the part of P read so far
3. Next character c: among those rows, take the ones with c in L (the c right before the part read so far); LF-mapping sends them to the rows that start with c + the part read so far

One LF step per character: m steps, **independent of** n.

⚠ Careful

Right to left, because LF-mapping goes one character *back* in T.

Backward search on the toy text

	F	*L*
0	$ GATTACATT	A
1	A $GATTACAT	T
2	A CATTA$GAT	T
3	A TTA$GATTA	C
4	A TTACATTA$	G
5	C ATTA$GATT	A
6	G ATTACATTA	$
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A
10	T TACATTA$G	A

all rows: $[0, 10]$

Search `TTA`, right to left:

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	L
0	$ GATTACATT	A_1
1	A $GATTACAT	T
2	A CATTA$GAT	T
3	A TTA$GATTA	C
4	A TTACATTA$	G
5	C ATTA$GATT	A_2
6	G ATTACATTA	$
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A_3
10	T TACATTA$G	A_4

A in $L[0..10]$: A_1, A_2, A_3, A_4

Search TTA, right to left:

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	L
0	$ GATTACATT	A
1	A_1 $GATTACAT	T
2	A_2 CATTA$GAT	T
3	A_3 TTA$GATTA	C
4	A_4 TTACATTA$	G
5	C ATTA$GATT	A
6	G ATTACATTA	$
7	T A$GATTACA	T
8	T ACATTA$GA	T
9	T TA$GATTAC	A
10	T TACATTA$G	A

A: $[1, 4]$

Search TTA, right to left:

- A: $[1, 4]$

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	L	
0	$ GATTACATT	A	
1	A $GATTACAT	T_1	T in $L[1..4]$: T_1, T_2
2	A CATTA$GAT	T_2	
3	A TTA$GATTA	C	
4	A TTACATTA$	G	
5	C ATTA$GATT	A	
6	G ATTACATTA	$	
7	T A$GATTACA	T	
8	T ACATTA$GA	T	
9	T TA$GATTAC	A	
10	T TACATTA$G	A	

Search TTA, right to left:

- A: $[1, 4]$

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	*L*	
0	$ GATTACATT	A	
1	A $GATTACAT	T	
2	A CATTA$GAT	T	
3	A TTA$GATTA	C	
4	A TTACATTA$	G	
5	C ATTA$GATT	A	
6	G ATTACATTA	$	
7	T_1A$GATTACA	T	TA: [7, 8]
8	T_2ACATTA$GA	T	
9	T TA$GATTAC	A	
10	T TACATTA$G	A	

Search TTA, right to left:

- A: [1, 4]
- TA: [7, 8]

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	L	
0	$ GATTACATT	A	
1	A $GATTACAT	T	
2	A CATTA$GAT	T	
3	A TTA$GATTA	C	
4	A TTACATTA$	G	
5	C ATTA$GATT	A	
6	G ATTACATTA	$	
7	T A$GATTACA	T_3	T in $L[7..8]$: T_3, T_4
8	T ACATTA$GA	T_4	
9	T TA$GATTAC	A	
10	T TACATTA$G	A	

Search TTA, right to left:

- A: $[1, 4]$
- TA: $[7, 8]$

Your turn

Before each step: which rows come next?

Backward search on the toy text

	F	L	
0	$ GATTACATT	A	
1	A $GATTACAT	T	
2	A CATTA$GAT	T	
3	A TTA$GATTA	C	
4	A TTACATTA$	G	
5	C ATTA$GATT	A	
6	G ATTACATTA	$	
7	T A$GATTACA	T	
8	T ACATTA$GA	T	
9	T_3TA$GATTAC	A	TTA: [9, 10]
10	T_4TACATTA$G	A	

Search TTA, right to left:

- A: [1, 4]
- TA: [7, 8]
- TTA: [9, 10]

Size 2: two occurrences.

Your turn

Before each step: which rows come next?

C and Occ

row i	$L[i]$	$	A	C	G	T
0	A	0	1	0	0	0
1	T	0	1	0	0	1
2	T	0	1	0	0	2
3	C	0	1	1	0	2
4	G	0	1	1	1	2
5	A	0	2	1	1	2
6	$	1	2	1	1	2
7	T	1	2	1	1	3
8	T	1	2	1	1	4
9	A	1	3	1	1	4
10	A	1	4	1	1	4
$C[c]$		0	1	5	6	7

$Occ(c, i)$ = number of c in $L[0..i]$

One step for character c, interval $[sp, ep]$:

$$sp \leftarrow C[c] + Occ(c, sp - 1)$$
$$ep \leftarrow C[c] + Occ(c, ep) - 1$$

Start: $[0, n-1]$. Not found: $sp > ep$.

Your turn 3 min

Check the step A → TA: from $[1, 4]$ to?

C and Occ

row i	$L[i]$	$	A	C	G	T
0	A	0	1	0	0	0
1	T	0	1	0	0	1
2	T	0	1	0	0	2
3	C	0	1	1	0	2
4	G	0	1	1	1	2
5	A	0	2	1	1	2
6	$	1	2	1	1	2
7	T	1	2	1	1	3
8	T	1	2	1	1	4
9	A	1	3	1	1	4
10	A	1	4	1	1	4
$C[c]$		0	1	5	6	7

$Occ(c, i)$ = number of c in $L[0..i]$

One step for character c, interval $[sp, ep]$:

$$sp \leftarrow C[c] + Occ(c, sp - 1)$$
$$ep \leftarrow C[c] + Occ(c, ep) - 1$$

Start: $[0, n-1]$. Not found: $sp > ep$.

Your turn 3 min

Check the step A → TA: from $[1, 4]$ to?

$sp = 7 + Occ(\mathtt{T}, 0) = 7,$
$ep = 7 + Occ(\mathtt{T}, 4) - 1 = 8$

Where are the matches?

row	L	suffix	SA stored?	
0	A	$	–	
1	T	A$	–	
2	T	ACATTA$	**4**	step 3
3	C	ATTA$	–	step 1
4	G	ATTACATTA$	–	
5	A	CATTA$	–	step 2
6	$	GATTACATTA$	**0**	
7	T	TA$	**8**	
8	T	TACATTA$	–	
9	A	TTA$	–	step 0
10	A	TTACATTA$	–	

The interval gives **rows**, not positions in T.

- Sampled suffix array: keep $SA[r]$ only where it is a multiple of s (here $s = 4$; bwa: 32)
- Row 9 is not sampled: walk LF, $9 \to 3 \to 5 \to 2$; row 2 has $SA = 4$
- each LF step is one position back: $4 + 3 = 7$

Why it needs so little memory

What the FM-index stores, for the human genome ($n = 3.1 \cdot 10^9$):

part	how	size
L	2 bits per base	0.8 GB
Occ	counts only every 128 rows (checkpoints), count the rest in L	small
C	5 numbers	nothing
SA	every 32nd value	0.4 GB

- A few GB in total (bwa, GRCh38, both strands: about 5 GB)
- Suffix array: $\sim$15 GB; suffix tree: 30–60 GB

Indices compared

index	query time	memory (human)	patterns
none (scan)	$O(nm)$	3 GB (the text)	any
k-mer table	$O(1)$ + hits	$\sim$12 GB	length k only
suffix tree	$O(m)$	30–60 GB	any
suffix array	$O(m \log n)$	$\sim$15 GB	any
FM-index	$O(m)$	a few GB	any

- bwa, bowtie2: FM-index
- segemehl: enhanced suffix array
- long-read mappers (minimap2): k-mer table of selected k-mers (minimisers)

Check your understanding

Your turn 5 min

1. Why does backward search take the same time for a bacterial and a human genome?
2. The interval becomes empty after three characters. What does that tell you?
3. Why does the FM-index not store the column F?
4. What does sampling the suffix array cost?

Reads do not match exactly

A read differs from the reference by

- sequencing errors (about 1 in 1000 bases for short reads)
- true differences: SNPs, small insertions and deletions

For 150 bp, at least one difference is common. Backward search for the whole read then ends in an empty interval.

Your turn 2 min

Two ideas to still find the read with an exact index?

Reads do not match exactly

A read differs from the reference by

- sequencing errors (about 1 in 1000 bases for short reads)
- true differences: SNPs, small insertions and deletions

For 150 bp, at least one difference is common. Backward search for the whole read then ends in an empty interval.

Your turn 2 min

Two ideas to still find the read with an exact index?

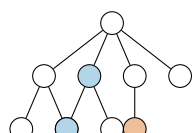
(1) allow mismatches during the search (next frame); (2) search short exact pieces and align around them (seed and extend).

Backtracking

- In backward search, try **all four characters** instead of only the read's, and count the mismatches
- Stop a branch when it has more than k mismatches, or when its interval is empty

Number of branches grows like $(3m)^k$:

- $k \leq 2$, short reads: fine
- indels, long reads, many errors: no



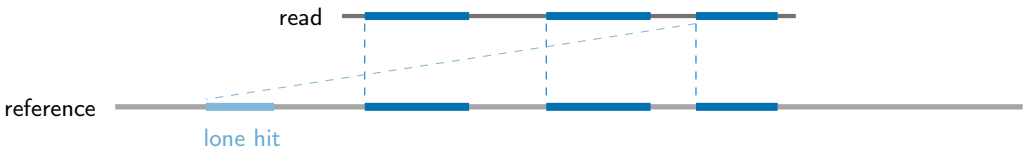
one level per character of P

Seed and extend



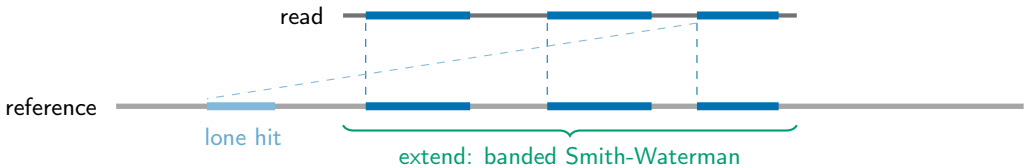
1. **Seeds**: long exact matches of read pieces, found with the FM-index

Seed and extend



1. **Seeds**: long exact matches of read pieces, found with the FM-index
2. **Chain**: keep seeds that lie on (nearly) the same diagonal, in order; drop lone hits

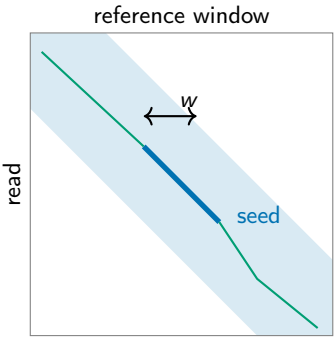
Seed and extend



1. **Seeds**: long exact matches of read pieces, found with the FM-index
2. **Chain**: keep seeds that lie on (nearly) the same diagonal, in order; drop lone hits
3. **Extend**: align the read around the chain with Smith-Waterman, restricted to a band

Lecture 2 (seeds, as in BLAST) + lecture 1 (local DP).

Banded alignment

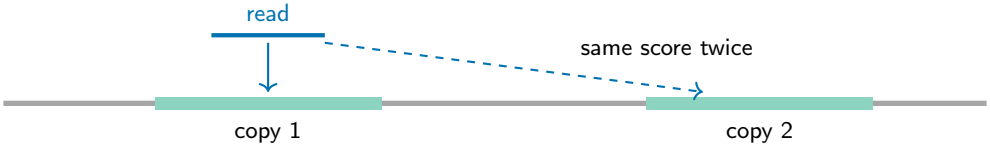


- The seed fixes the diagonal: the alignment stays close to it
- Fill only cells within distance w of the diagonal
- Cost $O(m\,w)$ instead of $O(m\,n)$

⚠ Careful

An indel longer than w does not fit into the band: the read is clipped or misaligned.

Where does this read belong?



- The read is shorter than the repeat and lies inside it: **both places align equally well**
- The mapper reports one of them as the alignment, **chosen at random**, the other as secondary (or only as a tag)
- Supplementary alignment: one read split over two places (e.g. across a rearrangement)

Your turn 2 min

The read is placed somewhere anyway. Where does a downstream tool learn that it is unsure?

Mapping quality: the definition

Definition: MAPQ (SAM specification)

$$MAPQ = -10 \log_{10} P(\text{mapping position is wrong})$$

rounded to an integer; 255 = not available.

- The Phred scale again: MAPQ 20 = 1 in 100 placements wrong, MAPQ 0 = no idea
- A statement about the **position**, not about the bases

What does the score mean? lecture 6

P(wrong) needs the alternatives: the read could come from any other place in the genome. A posterior over positions, normalised over all places the read could come from.

Mapping quality: what mappers compute

Nobody sums over all positions. Mappers **estimate** MAPQ heuristically.

bwa-mem, roughly:

- S_1: best alignment score (tag `AS`)
- S_2: second best (tag `XS`)
- $S_2 \geq S_1$: MAPQ $= 0$
- otherwise grows with $1 - S_2/S_1$ and with the length covered by seeds
- capped at 60

bowtie2: different formula, at most 42

Careful

"MAPQ ≥ 30" means different things for bwa and bowtie2. MAPQ values are not comparable across mappers.

Reference bias

- A read with the **reference** allele: one mismatch fewer than a read with the alternative allele
- Near other differences or in hard regions, the alternative reads score lower, get lower MAPQ, or do not map at all
- Variant callers then see fewer alternative reads than the sample has

⚠ Careful

Mapping against one reference is an assumption about the sample: that it looks like the reference.

One way out: pangenome graphs, references that contain many genomes and their variants.

Mappers compared (optional)

mapper	index	reads	spliced?
bwa-mem	FM-index	short (Illumina)	no
bowtie2	FM-index	short	no
segemehl	enhanced suffix array	short, many mismatches/indels	yes (split reads)
STAR	suffix array	RNA-seq	yes
HISAT2	graph FM-index	RNA-seq	yes
minimap2	minimisers (k-mer table)	long (and short)	yes

A SAM record

One line per alignment, eleven tab-separated columns, then optional tags.

1	QNAME	`sim-388388`	read name
2	FLAG	`147`	bits, later today
3	RNAME	`NC_000913.3`	reference sequence
4	POS	`21985`	leftmost position, 1-based
5	MAPQ	`60`	mapping quality
6	CIGAR	`42M1I107M`	the alignment, compressed
7	RNEXT	`=`	reference of the mate (=: same)
8	PNEXT	`21701`	position of the mate
9	TLEN	`-433`	insert size, signed
10	SEQ	`GCGTGTAGTCCA...` (150)	the read, as on the + strand
11	QUAL	`C=CGGCGGCGCG...` (150)	qualities, as in FASTQ
	tags	`NM:i:3 AS:i:132 XS:i:0`	also MD, MC, MQ, RG

One read of the exercise pairs (bwa mem on E. coli K-12 MG1655), SEQ and QUAL shortened.

The header

```
@HD VN:1.5  SO:coordinate
@SQ SN:NC_000913.3  LN:4641652
@RG ID:sim  SM:sim  PL:ILLUMINA
@PG ID:bwa  PN:bwa  VN:0.7.19-r1273  CL:bwa mem -t 4 -R
    @RG\tID:sim\tSM:sim\tPL:ILLUMINA ref.fa r1.fq.gz r2.fq.gz
@PG ID:samtools  PN:samtools  PP:bwa  VN:1.22.1  CL:samtools sort -@ 2 -o aln.bam -
```

Output of `samtools view -H aln.bam`, the exercise BAM

- `@SQ`: every reference sequence, with name and length (`SN:NC_000913.3` `LN:4641652`)
- `@RG`: read group: which sample, which run
- `@PG`: the program, its version and **the command line** that made the file; every tool adds one

The header is provenance inside the file.

FLAG: one number, twelve bits

bit	value	meaning
0	1	read is paired
1	2	proper pair
2	4	read unmapped
3	8	mate unmapped
4	16	read on reverse strand
5	32	mate on reverse strand

bit	value	meaning
6	64	first read of the pair
7	128	second read of the pair
8	256	secondary alignment
9	512	failed quality checks
10	1024	PCR or optical duplicate
11	2048	supplementary alignment

FLAG = sum of the values that apply. `samtools flags 99` decodes it.

Check: decode a FLAG

Your turn 4 min

Decode: 99, 147, 4. Which mate, which strand, mapped?

Check: decode a FLAG

Your turn 4 min

Decode: 99, 147, 4. Which mate, which strand, mapped?

- $99 = 64 + 32 + 2 + 1$: first of a proper pair, forward strand, mate on reverse

Check: decode a FLAG

Your turn 4 min

Decode: 99, 147, 4. Which mate, which strand, mapped?

- $99 = 64 + 32 + 2 + 1$: first of a proper pair, forward strand, mate on reverse
- $147 = 128 + 16 + 2 + 1$: second of a proper pair, reverse strand; the mate of a 99

Check: decode a FLAG

Your turn 4 min

Decode: 99, 147, 4. Which mate, which strand, mapped?

- $99 = 64 + 32 + 2 + 1$: first of a proper pair, forward strand, mate on reverse
- $147 = 128 + 16 + 2 + 1$: second of a proper pair, reverse strand; the mate of a 99
- 4: unmapped (and not paired: single-end)

Check: decode a FLAG

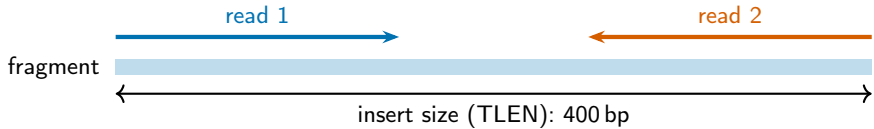
Your turn 4 min

Decode: 99, 147, 4. Which mate, which strand, mapped?

- $99 = 64 + 32 + 2 + 1$: first of a proper pair, forward strand, mate on reverse
- $147 = 128 + 16 + 2 + 1$: second of a proper pair, reverse strand; the mate of a 99
- 4: unmapped (and not paired: single-end)

If time: 83 and 163, the same pair the other way round.

Paired-end reads



- The two reads face each other; the mapper knows the typical insert size
- **Proper pair**: both mapped, expected orientation, expected distance
- Mate rescue: if one read is unique, its mate can be placed in a repeat nearby, if the insert reaches past the repeat

⚠ Careful: TLEN

Signed: positive for the leftmost read, negative for its mate. Its exact definition differed between tools.

CIGAR is the alignment from lecture 1

$POS = 101$, CIGAR = `3S8M1I6M2D5M`



`M`	aligned (match **or mismatch**)
`I`	insertion in the read
`D`	deletion from the read
`S`	soft clip: in SEQ, not aligned
`=/X`	match / mismatch (rarely used)

POS is the first **aligned** base: soft-clipped bases come before it.

Mismatches: in the tags (`NM:i:4` = 1 mismatch + 1 inserted + 2 deleted).

CIGAR is the alignment from lecture 1

$POS = 101$, CIGAR = `3S8M1I6M2D5M`

```
position     ↓101
reference    ATGAACTG-GAGTCTACGATGA
read      acgATGAAATGAGAGTCT--GATGA
```

`M`	aligned (match **or mismatch**)
`I`	insertion in the read
`D`	deletion from the read
`S`	soft clip: in SEQ, not aligned
`=/X`	match / mismatch (rarely used)

POS is the first **aligned** base: soft-clipped bases come before it.

Mismatches: in the tags (`NM:i:4` = 1 mismatch + 1 inserted + 2 deleted).

CIGAR is the alignment from lecture 1

$POS = 101$, CIGAR = `3S8M1I6M2D5M`

```
position    ↓101
reference    ATGAACTG-GAGTCTACGATGA
read      acgATGAAATGAGAGTCT--GATGA
CIGAR     SSSMMMMMMMMMIMMMMMMDDMMMMM
```

`M`	aligned (match **or mismatch**)
`I`	insertion in the read
`D`	deletion from the read
`S`	soft clip: in SEQ, not aligned
`=/X`	match / mismatch (rarely used)

POS is the first **aligned** base: soft-clipped bases come before it.

Mismatches: in the tags (`NM:i:4` = 1 mismatch + 1 inserted + 2 deleted).

Check: CIGAR

Your turn 4 min

A read has $POS = 2000$ and CIGAR `5S40M2I50M3D53M`.

1. How long is the read?
2. Where does it end on the reference?
3. Which bases are soft-clipped, and why might that be?

Check: CIGAR

Your turn 4 min

A read has $POS = 2000$ and CIGAR `5S40M2I50M3D53M`.

1. How long is the read?
2. Where does it end on the reference?
3. Which bases are soft-clipped, and why might that be?

- `M`, `I`, `S` consume the read: $5 + 40 + 2 + 50 + 53 = 150$

Check: CIGAR

Your turn 4 min

A read has $POS = 2000$ and CIGAR `5S40M2I50M3D53M`.

1. How long is the read?
2. Where does it end on the reference?
3. Which bases are soft-clipped, and why might that be?

- `M`, `I`, `S` consume the read: $5 + 40 + 2 + 50 + 53 = 150$
- `M`, `D` consume the reference: $40 + 50 + 3 + 53 = 146$, so it ends at $2000 + 146 - 1 = 2145$

Check: CIGAR

Your turn 4 min

A read has $POS = 2000$ and CIGAR `5S40M2I50M3D53M`.

1. How long is the read?
2. Where does it end on the reference?
3. Which bases are soft-clipped, and why might that be?

- `M`, `I`, `S` consume the read: $5 + 40 + 2 + 50 + 53 = 150$
- `M`, `D` consume the reference: $40 + 50 + 3 + 53 = 146$, so it ends at $2000 + 146 - 1 = 2145$
- the first 5 bases: e.g. low quality or adapter at the start, or a breakpoint

BAM: sorted and indexed

- **BAM**: the same records, binary and compressed (in blocks)
- **Sort by coordinate**: records ordered by reference and POS
- **Index** (`.bai`): which block holds which region

```
samtools sort -o aln.bam aln.sam
samtools index aln.bam
samtools view aln.bam NC_000913.3:100000-101000
```

The last command reads only the blocks of this region, not the whole file.

Your turn 2 min

Who needs to jump to a region?

BAM: sorted and indexed

- **BAM**: the same records, binary and compressed (in blocks)
- **Sort by coordinate**: records ordered by reference and POS
- **Index** (`.bai`): which block holds which region

```
samtools sort -o aln.bam aln.sam
samtools index aln.bam
samtools view aln.bam NC_000913.3:100000-101000
```

The last command reads only the blocks of this region, not the whole file.

Your turn 2 min

Who needs to jump to a region?

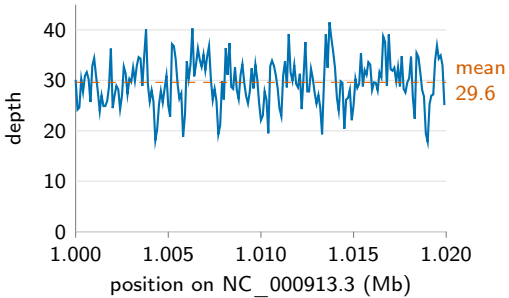
Genome browsers (show one region), variant callers (work region by region, in parallel).

Depth and duplicates

Depth: number of reads covering a position (`samtools depth`)

$$\text{Expected mean depth} = \frac{N \cdot \ell}{G}$$

(N reads of length ℓ, genome size G)



Exercise pairs, mean per 100 bp, 20 kb of a unique region

Duplicates: pairs with the same start positions and strands

- most likely PCR copies of one molecule
- marked (FLAG 1024), not deleted
- they would count the same evidence twice, e.g. for a variant

Depth comes back three times next week: variants, expression, assembly.

Check after every step

A step can fail silently: the tool exits without error, the file exists, the content is wrong.

after	**look at**	**plausible?**
sequencing	FastQC	quality, adapters, GC as expected for this library
trimming	FastQC again, trimmer report	adapters gone; how many reads and pairs were lost?
mapping	`samtools flagstat`, `samtools stats`	mapped and properly paired: close to all, for the right reference; insert sizes as in the library
depth	`samtools depth`	mean close to $N \cdot \ell / G$; gaps and peaks explained

Write down what you expect before you look.

Your turn 2 min

Only 60 % of the reads map to MG1655. What could have gone wrong?

Check after every step

A step can fail silently: the tool exits without error, the file exists, the content is wrong.

after	look at	plausible?
sequencing	FastQC	quality, adapters, GC as expected for this library
trimming	FastQC again, trimmer report	adapters gone; how many reads and pairs were lost?
mapping	`samtools flagstat`, `samtools stats`	mapped and properly paired: close to all, for the right reference; insert sizes as in the library
depth	`samtools depth`	mean close to $N \cdot \ell / G$; gaps and peaks explained

Write down what you expect before you look.

Your turn 2 min

Only 60 % of the reads map to MG1655. What could have gone wrong?

Wrong reference (another strain or species), contamination, adapters not trimmed, a mix-up of files. Find out before you go on: every later step inherits the problem.

“I ran bwa” is not a method

Your turn 3 min

Your colleague mapped reads last year. What do you need to get exactly her result again?

“I ran bwa” is not a method

Your turn 3 min

Your colleague mapped reads last year. What do you need to get exactly her result again?

- **input data**: accession (e.g. SRA run), or checksums of the files
- **reference**: name, version, source, checksum
- **tools**: names and exact versions
- **parameters**: every option, also the defaults that changed between versions
- **random seeds** (simulations, subsampling)
- **the order of the steps**: a script, not a description

Same name, different reference

- **GRCh37 vs. GRCh38**: the same gene at different coordinates
  BRCA1: chr17:41,196,312 (GRCh37) vs. chr17:43,044,295 (GRCh38)
- **Naming**: `chr1` (UCSC) vs. `1` (Ensembl)
- **Accession with version**: `NC_000913.3` is one exact sequence of *E. coli* K-12 MG1655; `.2` was a different one

⚠ Careful

Tools often do not fail on a mismatch: a region `chr1:1000-2000` on a reference that calls it `1` returns **nothing**, without an error.

A pinned environment

```
name: seqgen-ngs
channels:
  - conda-forge
  - bioconda
  - nodefaults
dependencies:
  - python=3.12
  - numpy=2.5
  - scipy=1.18
  - matplotlib=3.11
  - fastqc=0.13.0
  - fastp=1.1.0
  - bwa=0.7.19
  - htslib=1.22.1
  - samtools=1.22.1
  - bcftools=1.22
  - snakemake-minimal=9.27.1
  - megahit=1.2.9
  - segemehl=0.3.4
```

The environment of exercise 6 (`seqgen-ngs.yml`):

- channels: where the packages come from
- every tool with an **exact version**
- `check_env.sh` prints the versions: they go into your notes

One environment per project: another project can need other versions.

From shell script to workflow

`run.sh`

```
bwa index ref.fa
bwa mem ref.fa r1.fq r2.fq \
  | samtools sort -o aln.bam
samtools index aln.bam
```

runs everything, every time, in this order

`Snakefile`

```
rule map:
    input:
        ref="ref.fa", idx="ref.fa.bwt",
        r1="r1.fq", r2="r2.fq"
    output: "aln.bam"
    shell:
        "bwa mem {input.ref} {input.r1} {input.r2}"
        " | samtools sort -o {output}"
```

- each rule: inputs → outputs
- Snakemake builds the dependency graph (DAG) and **reruns only what is out of date**

Other workflow managers: Nextflow (same idea).

Take-home

1. Billions of reads: alignment needs an index, not a faster DP
2. $Q = -10 \log_{10} P_{\text{error}}$; expected errors add up the P, not the Q
3. QC shows symptoms; the cause is in the lab; thresholds are conventions; check again after every step
4. FM-index: BWT + LF-mapping, backward search in m steps, a few GB for a human genome
5. Mapping = exact seeds + banded DP; MAPQ: how sure the *position* is, estimated differently by each mapper
6. SAM: FLAG bits, CIGAR = the alignment; BAM sorted + indexed
7. Record data, reference, versions, parameters, seeds: a script, ideally a workflow

Next week

- Lecture 7: from the BAM to biology
  - variants (base qualities and MAPQ come back)
  - RNA-seq: spliced mapping, expression
  - genome assembly: what to do without a reference
  - ...
- Exercise 6: QC, mapping, a script, a workflow

⚠ Careful

Keep the BAM of exercise 6: exercise 7 might start from it.

Backup: inverting the BWT

step	**row**	L[row]	**text so far**
1	0	A	A$
2	1	T	TA$
3	7	T	TTA$
4	9	A	ATTA$
5	3	C	CATTA$
6	5	A	ACATTA$
7	2	T	TACATTA$
8	8	T	TTACATTA$
9	10	A	ATTACATTA$
10	4	G	GATTACATTA$

$L =$ `ATTCGA$TTAA` alone gives back T:

- start at row 0 (the suffix \$): $L[0]$ is the last letter of T
- LF-mapping: the row of the suffix one character longer
- each step prepends L[row]

Uses only L and C (and Occ).