

Introduction to NGS Data Analysis

Dr. Robert Kofler

September 17, 2013

REQUIREMENTS

- ▶ the lecture:
<http://drrobertkofler.wikispaces.com/NGSDataAnalysis>
- ▶ the data:
<http://popoolation.googlecode.com/files/teaching-data.zip>
- ▶ FastQC
- ▶ PoPoolation: <http://code.google.com/p/popoolation/>
- ▶ bwa
- ▶ samtools
- ▶ Picard
- ▶ IGV

SEQUENCING TECHNOLOGIES

Illumina HiSeq



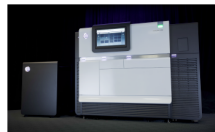
ABI Solid



Roche 454

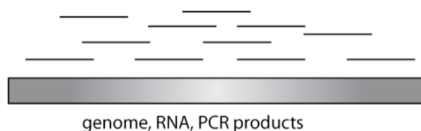


PacBio RS



UNIFYING FEATURE: SHORT READS

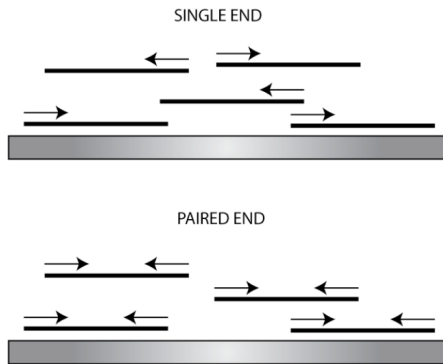
No matter the sequencing technology they all produce "short reads". These are stretches of DNA sequences randomly distributed in the feature of interest which is usually the genome or the transcriptome.



However the technologies vary with respect to

- ▶ read length (30 – 10000)
- ▶ number of reads (up to 100 million)
- ▶ error rate $< 1\% - 15\%$
- ▶ type of error (base substitutions, indels)

SINGLE END VS. PAIRED END READS



During "sample prep" (sample preparation) the genome (or the transcriptome) is randomly chopped into small pieces (e.g.: with a nebulizer).

If the resulting fragments (size 300-500bp) are sequenced only from one side we obtain single end reads. If they are sequenced from both ends we obtain paired end reads. In any case the reads are randomly distributed over the feature of interest.

Section 2

fastq

OUTPUT FROM NGS TECHNOLOGIES

So what do I get from such a NGS machine??

Not surprising the technologies have different output file formats. One common output is a "fastq" file. This name has historical reasons, it is derived from the first alignment software "fasta". The "q" stands for quality.

INSPECTING A FASTQ FILE

```

1 # download teaching-data.zip from
2 # http://popoolation.googlecode.com/files/
   teaching-data.zip
3 unzip teaching-data.zip
4 cd data
5 less read_1.fastq
6 # voila that's a fastq file

```

```

1 @HWUSI-EAS300R:7:1:7:674#0/1
2 CTTTTGTAGTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
3 +HWUSI-EAS300R:7:1:7:674#0/1
4 abbb_[V'a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
5 @HWUSI-EAS300R:7:1:9:1897#0/1
6 ACTTAATTATTTATGCTTTTCTCTACTCTGCACGGCATGCAAATGCAATATAGATGCAAGGCGAGCCGAAACAA
7 +HWUSI-EAS300R:7:1:9:1897#0/1
8 aaab`'bb^bababbaababbbbababbbbabaaaaabbbb[_bba_`^aabaabaaaa__aaaaa[Saa_`
9 @HWUSI-EAS300R:7:1:13:849#0/1
10 GAAATATTGCGTAGCCGAAACAAAAGAGTGCAAATACATTTTCGACGATGATGAGAGAGCTATTTCAGGGCTGTA
11 +HWUSI-EAS300R:7:1:13:849#0/1
12 a^`aaaabbb^`bab`a_Xaa_aa^_a__`_aaaa`ba`aaa`_`aaa`aZ_a^a]_^aR_a^_`'_`]^_`Y
13 .....

```





STRUCTURE OF A FASTQ-FILE

```
1 @HWUSI-EAS300R:7:1:7:674#0/1
2 CTTTGTAGTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
3 +HWUSI-EAS300R:7:1:7:674#0/1
4 abbb_[V'a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
```

Every read occupies four lines in a fastq file

- ▶ @: name of the read (always starts with '@')
- ▶ CTT..: the sequence of the read
- ▶ +: the name again (only useful with multiline fastq entries)
- ▶ the base quality of the DNA sequence; one character for every nucleotide

NAVIGATING A FASTQ-FILE

- ▶   scroll through the file
- ▶  next page
- ▶ 'b' previous page
- ▶ 'g' move to beginning of file
- ▶ 'G' move to end of file
- ▶ /blabla search for blabla
- ▶ 'n' move to the next search result
- ▶ 'N' move to the previous search result

EXERCISE: NAVIGATING A FASTQ-FILE

- ▶ search for the DNA sequence 'AAAAAAAAAAA' (10x). Is it usually repeated more than 10x?
- ▶ also search for the next three hits of 'AAAAAAAAAAA'
- ▶ search for the read 'HWUSI-EAS300R:7:1:14:1748'. How can it be found?
- ▶ what is the sequence of the last read in the file?

QUALITY DEMYSTIFIED

Basically every base quality entry gives you the probability that the corresponding DNA nucleotide is wrong (thus a sequencing error). So a quality of 0.001 means that on average 1 in 1000 bases having this quality is wrong.

The encoding of the quality involves a few steps

- ▶ get the decimal value of the quality character from an ASCII table (see next page)
- ▶ subtract the offset (sanger=33 illumina=64)
- ▶ voila: the quality of the base; usually a number between 0-40 [=dec(ascii)]
- ▶ if you want, you can go on and calculate the probability of being a sequencing error:

$$p = 10^{-\frac{\text{dec}(\text{ascii})}{10}}$$

ASCII TABLE

000	(nul)	016	► (dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	⊕ (soh)	017	◄ (dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	⊖ (stx)	018	↑ (dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥ (etx)	019	!! (dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦ (eot)	020	‡ (dc4)	036	\$	052	4	068	D	084	T	100	d	116	t
005	♣ (enq)	021	§ (nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠ (ack)	022	- (syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	• (bel)	023	‡ (etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	▣ (bs)	024	↑ (can)	040	(056	8	072	H	088	X	104	h	120	x
009	(tab)	025	↓ (em)	041)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂ (vt)	027	← (esc)	043	+	059	;	075	K	091	[107	k	123	{
012	♀ (np)	028	L (fs)	044	,	060	<	076	L	092	\	108	l	124	
013	(cr)	029	↔ (gs)	045	-	061	=	077	M	093]	109	m	125	}
014	♯ (so)	030	▲ (rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	⊗ (si)	031	▼ (us)	047	/	063	?	079	O	095	_	111	o	127	o

In informatics every character has a number. For the computer they are actually interchangeable. The binary '01000001' refers at the same time to the number '65' and to the character 'A'. Only the context decides which definition will be used. It is therefore fairly straight forward to translate numbers between 0-127 to characters.

EXAMPLE FASTQ DECODING

```

1 @read1
2 TTACGTTTTTT
3 +read1
4 87ba7777777

```

Base 'A' in Illumina encoding (*offset* = 64)

- ▶ A has the quality b
- ▶ b \Rightarrow 98 (from ascii table)
- ▶ $98 - 64 = 34$; my base quality for A is therefore 34
- ▶ error probability $p = 10^{-34/10} = 0.000398$

This error probability can be interpreted as 1 in 2511 ($= 1/0.000398$)
base pairs being wrongly sequenced (= sequencing error)

EXERCISES FASTQ DECODING

```
1 @read1
2 TTTACGTTTT
3 +read1
4 aaaa8aaaa
```

- ▶ Q1: Base quality of 'A' in Illumina
- ▶ Q2: Error probability of 'A' in Illumina
- ▶ Q3: Base quality of 'A' in Sanger
- ▶ Q4: Error probability of 'A' in Sanger
- ▶ Q5: Base quality of C in Sanger and Illumina. Is there anything weird about this results?
- ▶ Q6: Can you decide if the fastq file shown above is in Sanger or Illumina.

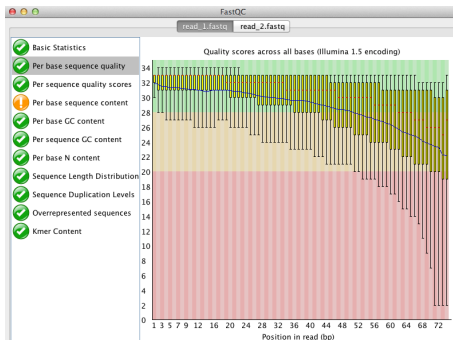
IS MY FASTQ-FILE SANGER OR ILLUMINA ENCODED?

If any of the bases has a negative quality with Illumina (64) than the encoding is Sanger (33). Therefore if you find any ascii character < 64 (e.g.: 1,2,3,4,5,6,7,8,9) in your fastq file than your encoding is Sanger; if not than it is Illumina.

Exercise: which quality encoding is your fastq file?

FASTQC: QUALITY CONTROL OF SHORT READS

After obtaining the short read files, the first questions you are usually asking is: How successful has the sequencing of my reads been? What is the quality of my reads? The user-friendly tool FastQC can help to answer these questions.



EXERCISE FASTQC

Open FastQC (double click in 'Applications' folder), load the file 'read_1.fastq' and answer the following questions:

- ▶ number of reads in your file
- ▶ the fastq encoding of your file
- ▶ the read length
- ▶ the average quality of your reads
- ▶ fraction of duplicates, triplicates, etc

Section 3

A quick single ended mapping experience

GREAT SHORT READS..

OK, great now we have the short reads, what can we do with them?
Any ideas?

SOME EXAMPLES OF APPLICATIONS FOR SHORT READS

- ▶ Where is the position of the read in the genome: mapping
- ▶ Can we piece them together into larger junks (like playing a puzzle): assembly
- ▶ Measure abundance of reads (count duplicates, triplicates etc)

Most biological interesting questions involve mapping of the reads. A few biological questions require assembly, and even less the counting of the reads (this is more for quality testing of the sequencing step). We will therefore only focus on mapping (alignment).

BEFORE MAPPING

ALWAYS make sure your data are complete. Quick and dirty

```

1 wc read_*
2 > 209288 209288 11279782 read_1.fastq
3 > 209288 209288 11279782 read_2.fastq
4 > 418576 418576 22559564 total

```

more professional

```

1 md5 read_*
2 >MD5 (read_1.fastq) = fd8fdfce336391e106fdc84ee60dd622
3 >MD5 (read_2.fastq) = 18d01db158ea29334d90b68880d9f6bb

```

The wordcount or the md5 sum should be compared to the values provided by the sequencing facility (e.g.: BGI).

MAPPING SOFTWARE

- ▶ BWA \Leftarrow
- ▶ Bowtie
- ▶ GEM
- ▶ BFAST
- ▶ STAMPY
- ▶ MAQ
- ▶ SOAP
- ▶ BLAT

In this introduction, we are using BWA for aligning short reads to the reference genome. BWA is currently one of the most widely used alignment algorithm. It is fast and flexible (many options)

FIRST: THE REFERENCE GENOME (FASTA-FILE)

Now in addition to the fastq file we are encountering the fasta-file. To view the reference genome type:

```
1 less dmel-2R-chromosome-r5.22.fasta
```

and you obtain:

```
1 >2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599; MD5=1589
  a9447d4dc94c048aa48ea5b8099d; length=21146708; release=r5.22; species=Dmel;
2 GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAAATTTGGTTAT
3 CTATTATAAAATGGATCCATATTTTAAAAATGTTAACAAAGGGTAAATGCGC
4 TTATACAAAGTATGAGGAAAGTTTGCGAAAGACTTCATAGCTTTGAAGAG
5 ....
```

- ▶ symbol indicating start of a new sequence '>' directly followed, without space, by the name of the sequence; after space some description of the sequence
- ▶ the actual sequence, mostly in chunks of 60 nucleotides per line (sometimes 50, the length is irrelevant)
- ▶ optional: additional sequences with the same formatting

PREPARING THE REFERENCE SEQUENCE FOR MAPPING

```

1 mkdir wg
2 awk '{print $1}' dmel-2R-chromosome-r5.22.fasta
   > wg/dmel-2R-short.fa
3 bwa index wg/dmel-2R-short.fa

```

Note; the awk command just prints the first column, where a column is defined as everything being separated by a space. With this command we are therefore removing the description of the fasta entry. This step is strongly recommended as unnecessarily long fasta identifiers may lead to problems in downstream analysis.

```

1 >2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599; MD5=1589
   a9447d4dc94c048aa48ea5b8099d; length=21146708; release=r5.22; species=Dmel;
2 GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAATTTGGTTAT
3 ...
4
5 => transformed into:
6
7 >2R
8 GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAATTTGGTTAT
9 ...

```

SINGLE END MAPPING

```

1 mkdir singleend
2 bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2
   wg/dmel-2R-short.fa read_1.fastq > singleend/read_1.
   sai
3 bwa samse wg/dmel-2R-short.fa singleend/read_1.sai read_1
   .fastq> singleend/read_1.sam

```

- ▶ -I input is in Illumina encoding (offset 64); do not provide this when input is in sanger! Very important parameter!
- ▶ -m not important; just telling bwa to process smaller amounts of reads at once
- ▶ -l 200 seed size (needs to be longer than the read length to disable seeding)
- ▶ -e 12 -d 12 gap length (for insertions and deletions)
- ▶ -o 1 maximum number of gaps
- ▶ -n 0.01 the number of allowed mismatches, in terms of probability. In general the lower the value the more mismatches are allowed. The exact translation is shown at the beginning of the mapping
- ▶ -t 2 number of threads, the more the faster

CONVERTING SAM TO BAM

- ▶ sam.. Sequence Alignment Map format \Rightarrow optimized for humans
- ▶ bam.. binary sam \Rightarrow optimized for computers

It is easily possible to convert a sam to bam and vice versa a bam to sam. In the following we convert a sam into a bam and finally sort the bam file

```
1 samtools view -Sb singleend/read_1.sam | samtools  
   sort - singleend/read_1
```

- ▶ -S input is sam
- ▶ -b output is bam (-S may be merged with -b to -Sb)
- ▶ 'sort - outfile' input for sorting is the pipe (rather than a file)

VISUALIZE THE RESULTS

First we need to index the bam file (necessary for visualization but nothing else)

```
1 samtools index singleend/read_1.bam
2 mkdir igv
```

- ▶ Open IGV (double click)
- ▶ Genomes \Rightarrow Create .genome File; Unique identifier="Dmel2R"; Descriptive name="Drosophila chromosome 2R"; FASTA file='wg/dmel-2R-short.fa';
- ▶ Save genome as 'igv/Dmel2R.genome'
- ▶ Load the mapped reads (singleend/read_1.bam)
- ▶ Zoom in at position 8,250,000
- ▶ Inspect the alignments

PAY SPECIAL ATTENTION TO

- ▶ the coverage graph on top
- ▶ SNPs (colored bases in the alignment); zoom in and out
- ▶ reference sequence (zoom in and out)
- ▶ alignment strands of the reads (find some forward and some reverse aligned reads);
- ▶ right click; color alignments by read strand
- ▶ find some insertions and deletions
- ▶ hover over read, what is happening now?
- ▶ find reads of low mapping quality (faint and white-ish)

INSPECTING THE SAM-FILE

```
1 less singleend/read_1.sam
```

and you will get something like

```
1 @SQ      SN:2R      LN:21146708
2 HWUSI-EAS300R:7:1:7:674#0 16 2R 7923527 37 74M * 0 0 TAACT... BAAC@B... XT:A:U NM:i:0
3 HWUSI-EAS300R:7:1:9:1897#0 0 2R 8172056 37 74M * 0 0 ACTTA... BBBC=A... XT:A:U NM:i:0
4 HWUSI-EAS300R:7:1:13:849#0 0 2R 8294036 37 74M * 0 0 GAAAT... B?ABBB... XT:A:U NM:i:1
```

- ▶ **samtools:** <http://samtools.sourceforge.net/>
- ▶ **documentation:** <http://samtools.sourceforge.net/SAMv1.pdf>

SAM FILE

A sam file has two parts, first a general info (lines starting with an '@') and second the aligned reads (all other lines) which have the following columns

- ▶ col 1: name of the read
- ▶ col 2: binary flag (contains lots of useful information, compressed into one number...)
- ▶ col 3: ID of the reference chromosome (e.g.: a read may align to 2L or 3R ...)
- ▶ col 4: position of the read in the reference chromosome (most 5' position is given)
- ▶ col 5: mapping quality (do not mix up with base quality)
- ▶ col 6: CIGAR string; specifies the alignment of the read with the reference chromosome
- ▶ col 10: sequence of the read (like in fastq)
- ▶ col 11: base quality of read (like in fastq); Per definition, the quality encoding has to be Sanger! (There should be only one version of sam file as opposed to the 5 fastq files)
- ▶ col 12+: optional information; dependent on the mapping software

SAM FILE; COLUMN 2 - BINARY FLAG

They developed a human readable format (sam) and a computer readable format (binary sam), yet they included a binary flag into the human readable format...seriously...wtf..

However, as the sam file is a currently quite standard we still have to learn the meaning of this flag. What you actually find in column 2 is an integer. In informatics every integer can be displayed as binary number:

2^6	2^5	2^4	2^3	2^2	2^1	2^0	
64	32	16	8	4	2	1	
0	0	0	0	1	0	1	=5
1	0	0	0	0	0	1	=65
0	0	1	1	0	0	0	=24
1	1	1	1	1	1	1	=127

In the binary flag of the sam-file, every of these 0 or 1 has a meaning. For example a 1 at the third position (**from the right**) means that the read could not be mapped to the reference genome (e.g.: to many mismatches) whereas a 0 at the third position means the read could be mapped to the reference genome

THE MOST IMPORTANT FLAGS

To make matters worse, the actual meaning of every flag is provided in hexadecimal numbers. Overview of the most important flags. For more details see <http://samtools.sourceforge.net/SAMv1.pdf>.

- ▶ 0x1 = first position from the right; if set to 1, than the read is a part of a paired end read (0 means single end)
- ▶ 0x2 = second position: if set to 1, than the reads map as a proper pair (definition depends on mapping software)
- ▶ 0x4 = third position: the read has not been mapped (remember 0 means mapped)
- ▶ 0x8 = fourth position: the mate (the other read of the pair) could not be mapped
- ▶ 0x10 = fifth position: the read maps as reverse complement (the sequence in col10 has been reverse complemented to)
- ▶ 0x20 = sixth position: the mate maps as reverse complement
- ▶ 0x100 = ninth position: this is a secondary alignment (thus there is an alternative more suitable alignment for the given read)
- ▶ 0x200 = tenth position: the read does not pass quality control (Illuminas requirements)
- ▶ 0x400 = eleventh position: the read is a duplicate (either PCR or optical); a suitable software has to be used to identify duplicates (e.g.: Picard)

QUICK WAY TO TEST FOR BINARY FLAGS?

We may for example find the integer 16 in our sam file. So what is the meaning of 16?

In my opinion one 'fairly' simple way to test for the presence of flags is to enter the Python command line (type 'python' in the command line and press enter). The binary and-operator `&` can be used to test whether the given bite is set to one. A number larger than zero means true (the bite is set) and zero means false.

- ▶ `16 & 0x1` the result is 0, which means that the read is single end
- ▶ `16 & 0x10` a result of 16 indicates that the read has been mapped as reverse complement
- ▶ `16 & 0x4` the result is 0, so the read has been mapped to the reference genome

Exercise: the first read in our sam file has the binary flag 16. Search for this read in the IGV. Is IGV displaying the read correctly (should be reverse complemented)?

EXERCISE BINARY FLAG

- ▶ interpret the binary flag '4'
- ▶ interpret the binary flag '0'
- ▶ you had a Unix introduction so what is the command at the bottom doing?
- ▶ type the command and interpret the results

```
1 cat read_1.sam | awk '{print $2}' | sort | uniq -c
```

EXERCISE: MENTALLY CONNECTING FASTQ AND SAM

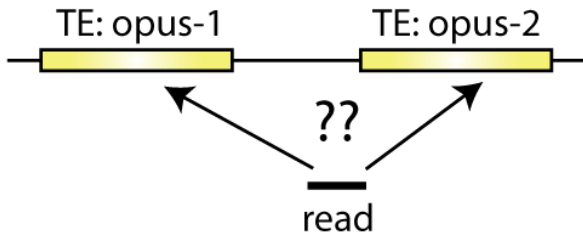
- ▶ Open the sam file (less)
- ▶ pick a read mapping to the forward strand
- ▶ search for this read in the fastq file (hint use less and the unique identifier of the read)
- ▶ anything notable?
- ▶ pick a read mapping to the reverse strand. Search for the read in the fastq file.
- ▶ anything notable?
- ▶ anything interesting about the quality encoding in the sam and the fastq file?

AMBIGUOUS MAPPING POSITION AND MAPPING QUALITY

Remember column 5 of the sam file contains the mapping quality. Similarly to the base quality, the mapping quality is the log scaled probability that the position of the read is wrong.

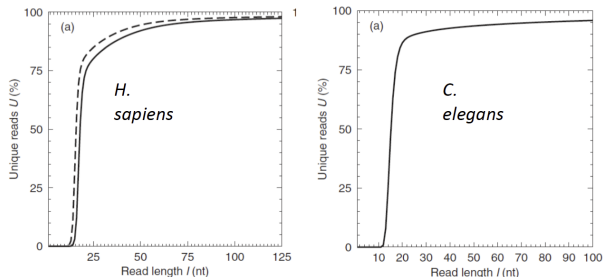
- ▶ 20.. one out of 100 reads is wrongly mapped
- ▶ 30.. one out of 1000 reads is wrongly mapped
- ▶ 0.. every read is wrongly mapped

A major cause for incorrect or ambiguous mapping positions are repetitive regions in the genome



FRACTION UNIQUE IN THE GENOME

Of course, the fraction of unique positions in the genome depends on the read length. Increasing the read length will lead to more unique alignment positions.



- ▶ mismatches are not considered in this study (thus when allowing for sequencing errors, longer reads will be required)
- ▶ in principle the same limitations also apply to paired end reads
- ▶ examples of repetitive regions: transposable elements, microsatellites, satellite DNA, copy number variation, large gene families

Source: Whiteford N. (2005) An analysis of feasibility of short read sequencing

REMOVE READS WITH A LOW MAPPING QUALITY

To get rid of these unreliably (ambiguously) aligned reads we can remove reads with a low mapping quality from the bam file. In the following we are removing all reads with a mapping quality lower than 20.

```
1 samtools view -q 20 -b singleend/read_1.bam >  
   singleend/read_1.q20.bam  
2 samtools index singleend/read_1.q20.bam
```

Exercise:

- ▶ load 'singleend/read_1.q20.bam into IGV
- ▶ compare to 'singleend/read_1.bam
- ▶ search for regions in the genome that differ between these two files

e.g.: 2R:8,251,975-8,254,865

CIGAR STRING (COL 6)

The CIGAR string ('col 6') only specifies the alignment between the read and the reference genome. Remember the starting position is given in 'col 4' and the sequence of the read in 'col 10'. Note mismatches are not affecting the CIGAR string

Read: TTAGATAAGATAGCTCTG

CIGAR: 18M

reference genome: AGCATGTTAGATAAGATAGCTGTGCTAGTA

aligned read: TTAGATAAGATAGCTCTG

Read: TTAGATAAGATAGGTG

CIGAR: 13M2D3M

reference genome: AGCATGTTAGATAAGATAGCTGTGCTAGTA

aligned read: TTAGATAAGATAG**GTG

Read: TTAGATAAAGGATACTG

CIGAR: 8M2I4M1D3M

reference genome: AGCATGTTAGATAA**GATAGCTGTGCTA

aligned read: TTAGATAAAGGATA*CTG

EXERCISE CIGAR

- ▶ open the sam file
- ▶ search for a read with an insertion (I) in the CIGAR string
- ▶ search for this read in the IGV. How is an insertion displayed in IGV? Is there any information missing in the alignment? What happens if you hover with the mouse over the insertion?
- ▶ repeat the same procedure for a deletion
- ▶ how is IGV displaying a deletion?
- ▶ are all reads aligning with this deletion actually having the same deletion?

```
1 I: HWUSI-EAS300R:7:1:49:1493#0 at 2R:7,952,751
2 D: HWUSI-EAS300R:7:1:1692:534#0 at 2R:7,996,071
```

CUSTOM ANALYSIS: GET ALL READS WITH DELETIONS

```
1 # filter all reads with deletions into a sam file
2 samtools view singleend/read_1.q20.bam | awk '$6~/D/' > singleend
  /deletions.sam
3
4 # convert the sam file into a bam file
5 # Note: we need to recreate the header of the sam file (-T)
6 samtools view -T wg/dmel-2R-short.fa -Sb singleend/deletions.sam |
  samtools sort - singleend/deletions
7 samtools index singleend/deletions.bam
```

Exercise:

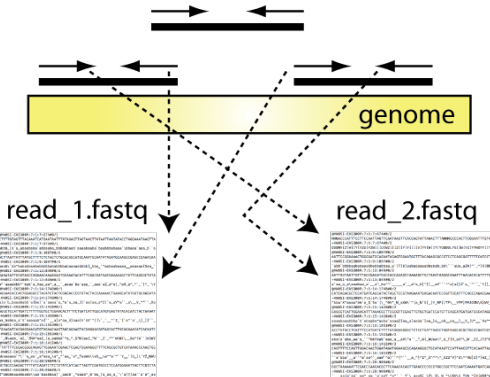
- ▶ load deletions.bam into the IGV
- ▶ repeat the whole procedure with insertions

Section 4

Pleasures of the full pipeline for paired end reads

PAIRED END READS

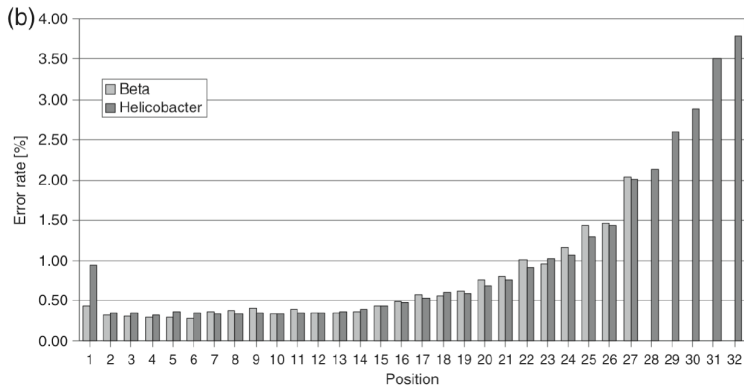
With the Illumina technology you will get two fastq-files for paired end reads. Reads are always provided in the same order in both fastq-files. The two reads of one pair can therefore be recognized by having the same index in the both fastq-files.



Note: the first read (read_1.fastq) is not necessarily the 5' read. Assignment as the first read is a stochastic process (therefore usually 50% 5' reads and 50% 3' reads).

TRIMMING OF READS; WHY?

Before we map the paired end reads we need to deal with a problem: Error rate of the reads is increasing with the length. Also remember your FastQC results, the base quality is decreasing with the length of the reads.



Source: Dohm J. (2008) Substantial biases in ultrashort read data sets from

TRIMMING OF READS; WHY?

A recent study clearly showed that trimming of the reads at low quality is the single quality filtering step that most dramatically improves the results (reduces analysis artefacts).

Minoche *et al.* *Genome Biology* 2011, **12**:R112
<http://genomebiology.com/2011/12/11/R112>



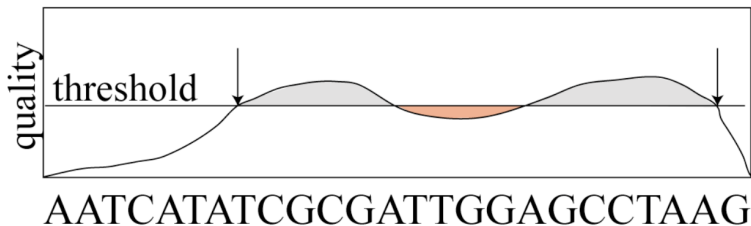
RESEARCH

Open Access

Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and Genome Analyzer systems

André E Minoche^{1,2}, Juliane C Dohm^{1,2} and Heinz Himmelbauer^{2*}

TRIMMING ALGORITHM OF POPOOLATION



- ▶ Given some arbitrary quality threshold (usually base quality of 20) the algorithm finds the highest scoring substring of the read.
- ▶ some fraction of the bases may be below the quality threshold, as long as new high score can be achieved.
- ▶ the algorithm is very similar to dynamic programming (Smith-Waterman)
- ▶ handles single end reads as well as paired end reads

TRIMMING

```

1 mkdir pairedend
2
3 # Trimming
4 perl <path-to-popoolation>/popoolation/basic-pipeline/trim-fastq.
    pl --input1 read_1.fastq --input2 read_2.fastq --min-length
    50 --no-5p-trim --quality-threshold 20 --fastq-type illumina
    --output pairedend/read_tr

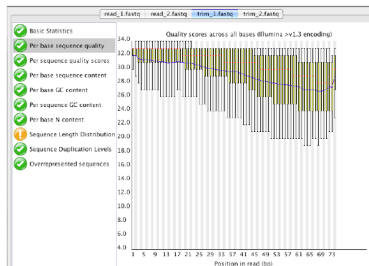
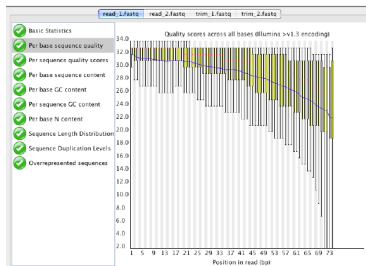
```

- ▶ `-input` the input files
- ▶ `-output` the output prefix; this will create three files with the extensions `.1` `.2` `_SE`;
- ▶ `-min-length` discard reads that are after trimming smaller than this threshold; Note this step may create orphan reads, i.e.: reads who lost their mate :(
- ▶ `-no-5p-trim` only trim reads at the 3' end; this is necessary for the removal of duplicates
- ▶ `-quality-threshold` reads should on average have a score higher than this threshold
- ▶ `-fastq-type` is the encoding of the base quality in sanger or illumina (remember offset)

TRIM STATISTIC

```
1 Read-pairs processed: 52322
2 Read-pairs trimmed in pairs: 52322
3 Read-pairs trimmed as singles: 0
4
5 FIRST READ STATISTICS
6 First reads passing: 52322
7 5p poly-N sequences trimmed: 0
8 3p poly-N sequences trimmed: 124
9 Reads discarded during 'remaining N filtering': 0
10 Reads discarded during length filtering: 0
11 Count sequences trimmed during quality filtering: 19928
12
13 Read length distribution first read
14 length count
15 50 322
16 51 327
17 52 351
18 53 359
19 54 358
20 55 381
21 56 366
```

EFFECT OF TRIMMING ON QUALITY



Exercise:

- ▶ open FastQC
- ▶ load read_1.fastq
- ▶ load read_tr.1
- ▶ compare the base quality between trimmed and untrimmed
- ▶ compare the sequence length distribution between trimmed and untrimmed

PAIRED END MAPPING

```
1 bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2 wg/dmel
  -2R-short.fa pairedend/read_tr_1 > pairedend/read_tr_1.sai
2 bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2 wg/dmel
  -2R-short.fa pairedend/read_tr_2 > pairedend/read_tr_2.sai
3 bwa sampe wg/dmel-2R-short.fa pairedend/read_tr_1.sai pairedend/
  read_tr_2.sai pairedend/read_tr_1 pairedend/read_tr_2 >
  pairedend/pe.sam
```

Note: the reads are actually mapped as single ends. Only the 'bwa sampe' step creates the paired end information.

PE FLAGS

Following some flags of the sam file that are mostly relevant for PE reads

- ▶ 0x1 = first position from the right; if set to 1, than the read is a part of a paired end read (0 means single end)
- ▶ 0x2 = second position: if set to 1, than the reads map as a proper pair (definition depends on mapping software)
- ▶ 0x4 = third position: the read has not been mapped (remember 0 means mapped)
- ▶ 0x8 = fourth position: the mate (the other read of the pair) could not be mapped
- ▶ 0x10 = fifth position: the read maps as reverse complement (the sequence in col10 has been reverse complemented to)
- ▶ 0x20 = sixth position: the mate maps as reverse complement

Exercise

- ▶ open 'pairedend/pe.sam'
- ▶ What is the meaning of the flag of the first read (83)?
- ▶ What is the meaning of the flag of the second read (163)?

SORTING WITH PICARD

We already sorted reads with samtools. Here we use Picard to sort reads, which is a bit more complicated. Sorting with Picard is necessary as otherwise the downstream analysis (MarkDuplicates) would not work

```
1 java -Xmx4g -jar <path-to-picard>/SortSam.jar I=pairedend/pe.sam
   O=pairedend/pe.sort.sam VALIDATION_STRINGENCY=SILENT SO=
   coordinate
```

- ▶ Picard runs with Java
- ▶ -Xmx4g give Java 4 Gb of memory
- ▶ -jar SortSam use the Java software SortSam
- ▶ I= input
- ▶ O= output
- ▶ SO= sort order; sort by coordinate
- ▶ VALIDATION_STRINGENCY= Picard is like a Princess that is constantly complaining about every small deviation of our sam file from the most stringent requirements. I have never found a sam file satisfying all of Picards demands ⇒ 'shut up Picard'

VISUALIZING PE READS

```
1 samtools view -Sb pe.sort.sam > pe.sort.bam
2 samtools index pe.sort.bam
```

- ▶ Load the mapped reads (pairedend/pe.sort.bam)
- ▶ Zoom in at position 8,250,000
- ▶ Right click on alignment ⇒ View as pairs
- ▶ Inspect the alignments
- ▶ Find some orphan reads (reads without mate). What could be the reason for orphan reads?

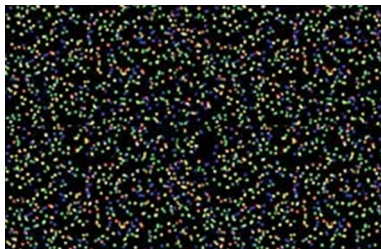
EXAMPLE OF PE READS IN IGV



DUPLICATES

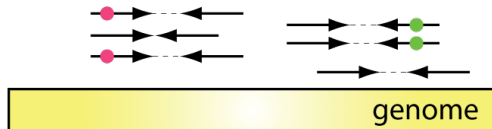
There are two sources of duplicates with the Illumina technology

- ▶ Optical duplicates: they are occurring during the sequencing step; The algorithm responsible for identifying isolated clusters wrongly identifies a single cluster as two (see picture below).
- ▶ PCR duplicates; occurs during sample preparation where a PCR step is necessary to amplify the amount of DNA for the sequencing.



HOW TO RECOGNIZE DUPLICATES

One common proxy is to use the mapping positions of PE reads to recognize duplicates where reads having exactly identical positions are marked as duplicates. This has the advantage that it also allows for sequencing errors within duplicated reads. The chances that two PE reads accidentally have identical positions are minimal.



Duplicates are marked by dots having identical colors.

DUPLICATES WITH TRIMMED READS

During trimming reads may be truncated at sequences having a low quality, thus duplicated reads may end up having different lengths. Fortunately, removal of duplicates can still be performed if only the 3' ends of reads are trimmed (remember that the quality deteriorates mostly at the 3' end of reads). This is because Picard recognizes duplicates by PE reads having identical 5' positions (5' of the read not the genome).



Duplicates are marked by dots having identical colors.

REMOVING DUPLICATES

```

1 # the following only accepts a sam file sorted by
   Picard.
2 java -Xmx4g -jar <path-to-picard>/MarkDuplicates.jar
   I=pairedend/pe.sort.sam O=pairedend/pe.sort.
   duprem.sam M=pairedend/pe.dupstat.txt
   VALIDATION_STRINGENCY=SILENT REMOVE_DUPLICATES=
   true
3 samtools index pe.sort.duprem.bam

```

- ▶ I= input file
- ▶ O= output file for reads
- ▶ M= output file of statistics (how many identified duplicates)
- ▶ REMOVE_DUPLICATES= remove duplicates from the output file rather than just marking them (remember flag in sam-file 0x400)

VISUALIZE THE RESULTS

```
1 samtools view -Sb pe.sort.duprem.sam > pe.sort.  
   duprem.bam  
2 samtools index pe.sort.duprem.bam
```

- ▶ Load the reads with duplicates (pairedend/pe.sort.bam)
- ▶ Load the reads without duplicates (pairedend/pe.sort.duprem.bam)
- ▶ Zoom in at position 8,250,000
- ▶ Compare the two files; Can you identify any duplicates that have been removed?

e.g.: 2R:8,172,161

REMOVE LOW QUALITY ALIGNMENTS

The following command ensures that we remove ambiguously mapped reads and only retain PE reads where both mates align with the reference genome

```
1 samtools view -q 20 -f 0x0002 -F 0x0004 -F 0x0008 -b  
   pe.sort.duprem.bam > pe.sort.duprem.mq20.bam
```

- ▶ -q 20 only keep reads with a mapping quality higher than 20 (remove ambiguously aligned reads)
- ▶ -f 0x0002 only keep proper pairs (remember flags from sam file)
- ▶ -F 0x0004 remove reads that are not mapped
- ▶ -F 0x0008 remove reads with an un-mapped mate

Note '-f' means only keep reads having the given flag and '-F' discard all reads having the given flag.

WHAT NOW?

Now that we successfully run the full pipeline for PE reads, what can we do with the resulting .bam files. Well move on to answer biologically relevant questions, of course!

For example, you may:

- ▶ use PoPoolation to identify regions of low nucleotide diversity in natural population, which could indicate regions that have been a positively selected
- ▶ use PoPoolation2 to identify differentiation between two or more populations.
- ▶ identify TE insertions in natural populations using PoPoolation TE
- ▶ run a RNAseq pipeline
- ▶ call variants (e.g.: SNPs with GATK)
- ▶

FINAL EXAM

- ▶ Repeat the whole paired end pipeline with the following modifications
- ▶ conduct the analysis in the folder pairedendexam
- ▶ trimming: quality threshold 20; minimum length 60
- ▶ mapping: allow for 2 gaps
- ▶ final filtering: mapping quality of 25
- ▶ how many reads do you get in the end? (Hint: wc)

Section 5

Solutions

SOLUTION NAVIGATION

- ▶ /AAAAAAAAAAA yes they are usually longer than 10 repeats
- ▶ n n n
- ▶ Go to beginning of file 'g' and search again (search is always down from current position)
- ▶ TTTGTTAAGTCATATGTTTTATTA...

EXERCISES FASTQ DECODING

- ▶ Q1: 33
- ▶ Q2: 0.0005011872
- ▶ Q3: 64
- ▶ Q4: $3.98e - 07$
- ▶ Q5: *sanger* = 23; *illumina* = -8; negative quality is nonsense
- ▶ Q6: yes, as the quality can not be negative the file is in sanger encoding