Subread/Rsubread Users Guide

Subread v1.5.0-p1/Rsubread v1.20.3

1 February 2016

Wei Shi and Yang Liao

Bioinformatics Division
The Walter and Eliza Hall Institute of Medical Research
The University of Melbourne
Melbourne, Australia

Contents

Chapter 1

Introduction

The Subread/Rsubread packages comprise a suite of high-performance software programs for processing next-generation sequencing data. Included in these packages are `Subread` aligner, `Subjunc` aligner, `Subindel` long indel detection program, `featureCounts` read quantification program, `exactSNP` SNP calling program and other utility programs. This document provides a detailed description to the programs included in the packages.

`Subread` and `Subjunc` aligners adopt a mapping paradigm called "seed-and-vote" [1]. This is an elegantly simple multi-seed strategy for mapping reads to a reference genome. This strategy chooses the mapped genomic location for the read directly from the seeds. It uses a relatively large number of short seeds (called subreads) extracted from each read and allows all the seeds to vote on the optimal location. When the read length is <160 bp, overlapping subreads are used. More conventional alignment algorithms are then used to fill in detailed mismatch and indel information between the subreads that make up the winning voting block. The strategy is fast because the overall genomic location has already been chosen before the detailed alignment is done. It is sensitive because no individual subread is required to map exactly, nor are individual subreads constrained to map close by other subreads. It is accurate because the final location must be supported by several different subreads. The strategy extends easily to find exon junctions, by locating reads that contain sets of subreads mapping to different exons of the same gene. It scales up efficiently for longer reads.

`Subread` is a general-purpose read aligner. It can be used to align reads generated from both genomic DNA sequencing and RNA sequencing technologies. It has been successfully used in a number of high-profile studies [2, 3, 4, 5, 6]. `Subjunc` is specifically designed to detect exon-exon junctions and to perform full alignments for RNA-seq reads. Note that `Subread` performs local alignments for RNA-seq reads, whereas `Subjunc` performs global alignments for RNA-seq reads. `Subread` and `Subjunc` comprise a read re-alignment step in which reads are re-aligned using genomic variation data and junction data collected from the initial mapping.

The `Subindel` program carries out local read assembly to discover long insertions and deletions. Read mapping should be performed before running this program.

The featureCounts program is designed to assign mapped reads or fragments (paired-end data) to genomic features such as genes, exons and promoters. It is a light-weight read counting program suitable for count both gDNA-seq and RNA-seq reads for genomic features[7]. The

Subread-featureCounts-limma/voom pipeline has been found to be one of the best-performing pipelines for the analyses of RNA-seq data by the SEquencing Quality Control (SEQC) study, the third stage of the well-known MicroArray Quality Control (MAQC) project [8].

Also included in this software suite is a very efficient SNP caller – ExactSNP. ExactSNP measures local background noise for each candidate SNP and then uses that information to accurately call SNPs.

These software programs support a variety of sequencing platforms including Illumina GA/HiSeq, ABI SOLiD, Life Science 454, Helicos Heliscope and Ion Torrent. They are released in two packages – SourceForge *Subread* package and Bioconductor *Rsubread* package.

Chapter 2

Preliminaries

2.1 Citation

If you use Subread or Subjunc aligners, please cite:

> Liao Y, Smyth GK and Shi W. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. Nucleic Acids Research, 41(10):e108, 2013
> http://www.ncbi.nlm.nih.gov/pubmed/23558742

If you use featureCounts, please cite:

> Liao Y, Smyth GK and Shi W. featureCounts: an efficient general-purpose program for assigning sequence reads to genomic features. Bioinformatics, 2013 Nov 30. [Epub ahead of print]
> http://www.ncbi.nlm.nih.gov/pubmed/24227677

2.2 Download and installation

2.2.1 SourceForge Subread package

Installation from a binary distribution

This is the easiest way to install the Subread package onto your computer. Download a Subread binary distribution that suits your oprating system, from the SourceForge website http://subread.sourceforge.net. The operating systems currently being supported include multiple variants of Linux (Debian, Ubuntu, Fedora and Cent OS) and Mac OS X. Both 64-bit and 32-bit machines are supported. The executables can be found in the 'bin' diretory of the binary package.

To install Subread package for other operating systems such as FreeBSD and Solaris, you will have to install them for the source.

Installation from the source package

Download Subread source package to your working directory from SourceForge http://subread.sourceforge.net, and type the following command to uncompress it:

```
tar zxvf subread-1.x.x.tar.gz
```

Enter `src` directory of the package and issue the following command to install it on a Linux operating system:

```
make -f Makefile.Linux
```

To install it on a Mac OS X operating system, issue the following command:

```
make -f Makefile.MacOS
```

To install it on a FreeBSD operating system, issue the following command:

```
make -f Makefile.FreeBSD
```

To install it on Oracle Solaris or OpenSolaris computer operating systems, issue the following command:

```
make -f Makefile.SunOS
```

To install it on a Windows computer, you will need to firstly install a unix-like environment such as cygwin and then install the Subread package.

A new directory called `bin` will be created under the home directory of the software package, and the executables generated from the compilation are saved to that directory. To enable easy access to these executables, you may copy them to a system directory such as `/usr/bin` or add the path to them to your search path (your search path is usually specified in the environment variable `'PATH'`).

2.2.2 Bioconductor Rsubread package

You have to get R installed on my computer to install this package. Lauch an R session and issue the following command to install it:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Rsubread")
```

Alternatively, you may download the Rsubread source package directly from http://bioconductor.org/packages/release/bioc/html/Rsubread.html and install it to your R from the source.

2.3 How to get help

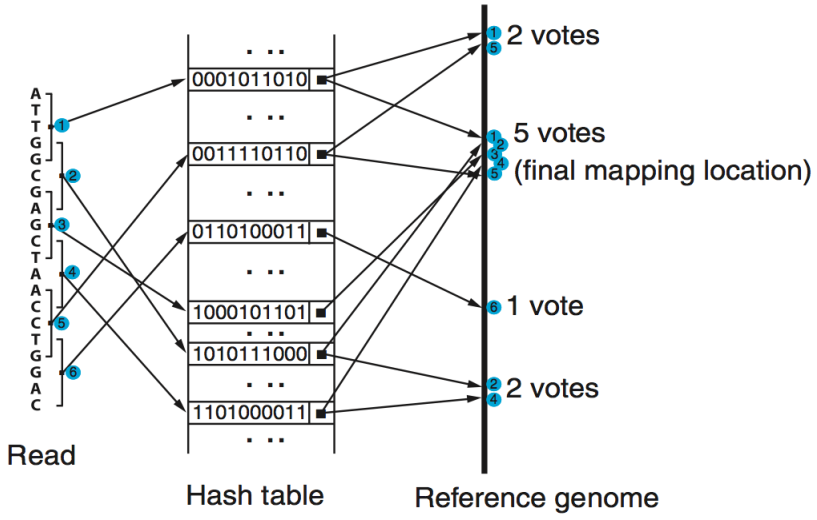
Bioconductor mailing list (`http://bioconductor.org/`) and SeqAnswer forum (`http://www.seqanswers.com`) are the best places to get help and to report bugs. Alternatively, you may contact Wei Shi (shi at wehi dot edu dot au) directly.

Chapter 3

The seed-and-vote mapping paradigm

3.1 Seed-and-vote

We have developed a new read mapping paradigm called "seed-and-vote" for efficient, accurate and scalable read mapping [1]. The seed-and-vote strategy uses a number of overlapping seeds from each read, called *subreads*. Instead of trying to pick the best seed, the strategy allows all the seeds to vote on the optimal location for the read. The algorithm then uses more conventional alignment algorithms to fill in detailed mismatch and indel information between the subreads that make up the winning voting block. The following figure illustrates the proposed seed-and-vote mapping approach with an toy example.

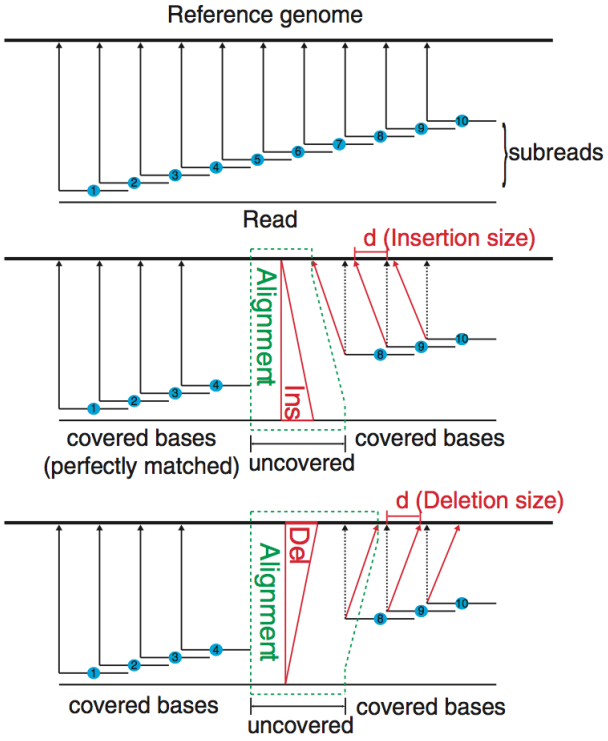


Two aligners have been developed under the seed-and-vote paradigm, including `Subread` and `Subjunc`. `Subread` is a general-purpose read aligner, which can be used to map both genomic DNA-seq and RNA-seq read data. Its running time is determined by the number of *subreads* extracted from each read, not by the read length. Thus it has an excellent maping scalability, ie. its running time has only very modest increase with the increase of read length.

`Subread` uses the largest mappable region in the read to determine its mapping location, therefore it automatically determines whether a global alignment or a local alignment should be found for the read. For the exon-spanning reads in a RNA-seq dataset, `Subread` performs local alignments for them to find the target regions in the reference genome that have the largest overlap with them. Note that `Subread` does not perform global alignments for the exon-spanning reads and it soft clips those read bases which could not be mapped. However, the `Subread` mapping result is sufficient for carrying out the gene-level expression analysis using RNA-seq data, because the mapped read bases can be reliably used to assign reads, including both exonic reads and exon-spanning reads, to genes.

To get the full alignments for exon-spanning RNA-seq reads, the `Subjunc` aligner can be used. `Subjunc` is designd to discover exon-exon junctions from using RNA-seq data, but it performs full alignments for all the reads at the same time. The `Subjunc` mapping results should be used for detecting genomic variations in RNA-seq data, allele-specific expression analysis and exon-level gene expression analysis. The Section 3.3 describes how exon-exon junctions are discovered and how exon-spanning reads are aligned using the seed-and-vote paradigm.

3.2 Detection of short indels



The seed-and-vote paradigm is very powerful in detecting short indels (insertions and deletions). The figure below shows how we use the *subreads* to confidently detect short indels. When there is an indel existing in a read, mapping locations of subreads extracted after the indel will be shifted to the left (insertion) or to the right (deletion), relative to the mapping

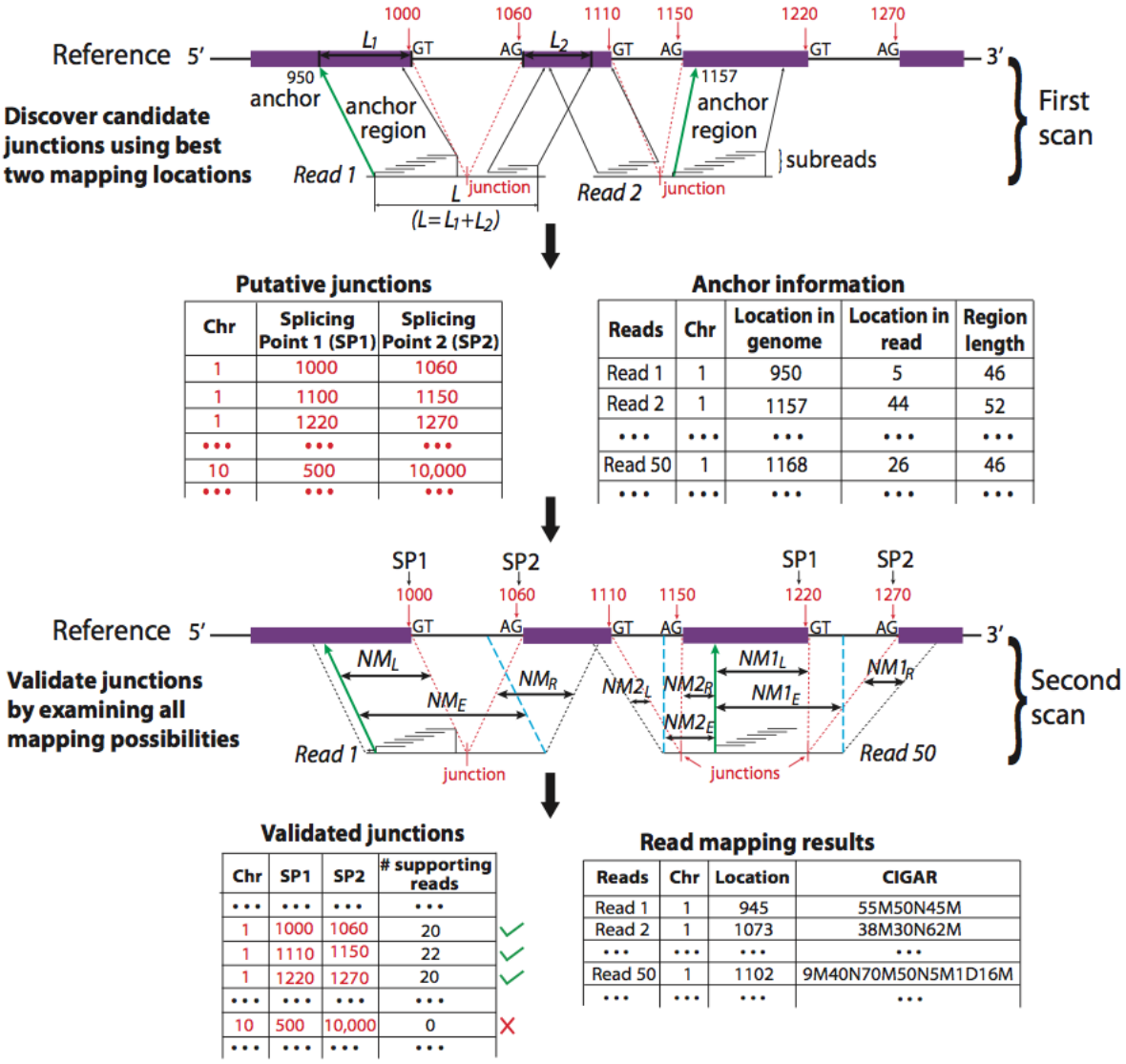
locations of subreads at the left side of the indel. Therefore, indels in the reads can be readily detected by examining the difference in mapping locations of the extracted subreads. Moreover, the number of bases by which the mapping location of subreads are shifted gives the precise length of the indel. Since no mismatches are allowed in the mapping of the subreads, the indels can be detected with a very high accuracy.

3.3 Detection of exon-exon junctions

The seed-and-vote paradigm is also very useful in detecting exon-exon junctions, because the short subreads extracted across the entire read can be used to detect short exons in a sensitive and accurate way. The figure below shows the schematic of detecting exon-exon junctions and mapping RNA-seq reads by `Subjunc`, which uses this paradigm.

The first scan detects all possible exon-exon junctions using the mapping locations of the subreads extracted from each read. Matched donor ('GT') and receptor ('AG') sites are required for calling junctions. Exons as short as 16bp can be detected in this step. The second scan verifies the putative exon-exon junctions discovered from the first scan by performing re-alignments for the junction reads. The output from `Subjunc` includes the list of verified junctions and also the mapping results for all the reads. Orientation of splicing sites is indicated by 'XA' tag in section of optional fields in mapping output.

By default, `Subjunc` only reports canonical exon-exon junctions it has discovered (ie. presence of donor ('GT') and receptor ('AG') sites is required). However, users may turn on '–allJunctions' option to instruct `Subjunc` to report all junctions including both canonical and non-canonical ones.



3.4 Detection of structural variants (SVs)

`Subread` and `Subjunc` can be used detect SV events including long indel, duplication, inversion and translocation, in RNA-seq and genomic DNA-seq data.

Detection of long indels is conducted by performing local read assembly. When the specified indel length ('-I' option in SourceForge `C` or 'indels' paradigm in `Rsubread`) is greater than 16, `Subread` and `Subjunc` will automatically start the read assembly process to detect long indels (up to 200bp).

Breakpoints detected from SV events will be saved to a text file ('.breakpoint.txt'), which includes chromosomal coordinates of breakpoints and also the number of reads supporting each pair of breakpoints found from the same SV event.

For the reads that were found to contain SV breakpoints, extra tags will be added for

them in mapping output. These tags include CC(chromosome name), CP(mapping position), CG(CIGAR string) and CT(strand), and they describe the secondary alignment of the read (the primary alignment is described in the main fields).

3.5 Two-scan read alignment

`Subread` and `Subjunc` aligners employ a two-scan approach for read mapping. In the first scan, the aligners use seed-and-vote method to identify candidate mapping locations for each read and also discover short indels, exon-exon junctions and structural variants. In the second scan, they carry out final alignment for each read using the variant and junction information. Variant and junction data (including chromosomal coordinates and number of supporting reads) will be output along with the read mapping results. To the best of our knowledge, `Subread` and `Subjunc` are the first to employ a two-scan mapping strategy to achieve a superior mapping accuracy. This strategy was later adopted by other aligners as well (called 'two-pass').

3.6 Multi-mapping reads

Multi-mapping reads are those reads that map to more than one genomic location with the same similarity score (eg. number of mis-mismatched bases). `Subread` and `Subjunc` aligners can effectively detect multi-mapping reads by closely examining candidate locations which receive the highest number of votes or second highest number of votes. Numbers of mis-matched bases and matched bases are counted for each candidate location during the final re-alignment step and they are used for identifying multi-mapping reads. For RNA-seq data, a read is called as a multi-mapping read if it has two or more candidate mapping locations that have the same number of mis-matched bases and this number is the smallest in all candidate locations being considered. For genomic DNA-seq data, a read is called as a multi-mapping read if it has two or more candidate locations that have the same number of matched bases and this number is the largest among all candidate locations being considered. Note that for both RNA-seq and genomic DNA-seq data, any alignment reported for a multi-mapping read must not have more than threshold number of mis-matched bases (as specified in '-M' parameter).

For the reporting of a multi-mapping read, users can choose to not report any alignment for the read ('-u' option) or report up to a pre-defined number of alignments ('-B' option).

3.7 Mapping of paired-end reads

For the mapping of paired-end reads, we use the following formula to obtain a list of candidate mapping locations for each read pair:

$$PE_{score} = w * (V_1 + V_2)$$

where V_1 and V_2 are the number of votes received from two reads from the same pair, respectively. w has a value of 1.3 if mapping locations of the two reads are within the nominal paired-end distance (or nominal fragment length), and has a value of 1 otherwise.

Up to 4,096 posssible alignments will be examined for each read pair and a maximum of three candidate alignments with the highest PE_{score} will be chosen for final re-alignment. Total number of matched bases (for genomic DNA-seq data) or mis-matched bases (for RNA-seq data) will be used to determine the best mapping in the final re-alignment step.

3.8 Recommended aligner setting

It is recommended to report uniquely mapped reads only when running `Subread` and `Subjunc` aligners since this will give the most accurate mapping result. By default, only uniquely mapped reads are reported when running aligners in Bioconductor Rsubread package. This however needs to be explicitly specified when running aligners in SourceForge Subread package (`-u`).

Chapter 4

Mapping reads generated by genomic DNA sequencing technologies

4.1 A quick start for using SourceForge **Subread** package

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Build an index

Build a base-space index (default). You can provide a list of FASTA files or a single FASTA file including all the reference sequences.

```
subread-buildindex -o my_index chr1.fa chr2.fa ...
```

Step 2: Align reads

Map single-end reads from a gzipped file using 5 threads and save mapping results to a BAM file:
`subread-align -T 5 -i my_index -r reads.txt.gz -o subread_results.bam`

Detect indels of up to 16bp:
`subread-align -I 16 -i my_index -r reads.txt -o subread_results.sam`

Report up to three best mapping locations:
`subread-align -B 3 -i my_index -r reads.txt -o subread_results.sam`

Report uniquely mapped reads only:
`subread-align -u -i my_index -r reads.txt -o subread_results.sam`

Map paired-end reads:
`subread-align -d 50 -D 600 -i my_index -r reads1.txt -R reads2.txt`

```
-o subread_results.sam
```

4.2 A quick start for using Bioconductor **Rsubread** package

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Building an index

To build the index, you must provide a single FASTA file (eg. "genome.fa") which includes all the reference sequences.

```
library(Rsubread)
buildindex(basename="my_index",reference="genome.fa")
```

Step 2: Aligning the reads

Map single-end reads using 5 threads:

```
align(index="my_index",readfile1="reads.txt.gz",output_file="rsubread.bam",nthreads=5)
```

Detect indels of up to 16bp:

```
align(index="my_index",readfile1="reads.txt.gz",output_file="rsubread.bam",indels=16)
```

Report up to three best mapping locations:

```
align(index="my_index",readfile1="reads.txt.gz",output_file="rsubread.bam",nBestLocations=3)
```

Map paired-end reads:

```
align(index="my_index",readfile1="reads1.txt.gz",readfile2="reads2.txt.gz",
output_file="rsubread.bam",minFragLength=50,maxFragLength=600)
```

4.3 Index building

The `subread-buildindex` (`buildindex` function in Rsubread) program builds an index for reference genome by creating a hash table in which keys are 16bp mers (subreads) extracted from the genome and values are their chromosomal locations. By default, subreads are extracted from the genome at a 2bp interval. The reference sequences should be in FASTA format (the header line for each chromosomal sequence starts with ">").

Table 1 describes the arguments used by the `subread-buildindex` program.

Table 1: Arguments used by the `subread-buildindex` program (`buildindex` function in Rsubread). Arguments in parenthesis in the first column are used by `buildindex`.

Arguments	Description
chr1.fa, chr2.fa, ... (`reference`)	Give names of chromosome files. Note that in Rsubread, only a single FASTA file including all reference sequences should be provided.
-B (`indexSplit=FALSE`)	Create one block of index. The built index will not be split into multiple pieces. This makes the largest amount of memory be requested when running alignments, but it enables the maximum mapping speed to be achieved. This option overrides -M when it is provided as well.
-c (`colorspace`)	Build a color-space index.
-f $< int >$ (`TH_subread`)	Specify the threshold for removing uninformative subreads (highly repetitive 16bp mers). Subreads will be excluded from the index if they occur more than threshold number of times in the reference genome. Default value is 100.
-F (`gappedIndex=FALSE`)	Build a full index for the reference genome. 16bp mers (subreads) will be extracted from every position of the reference genome. Under default setting ('-F' is not specified), subreads are extracted in every three bases from the genome.
-M $< int >$ (`memory`)	Specify the Size of requested memory(RAM) in megabytes, 8000MB by default. With the default value, the index built for a mammalian genome (eg. human or mouse genome) will be saved into one block, enabling the fastest mapping speed to be achieved. The amount of memory used is $\sim$ 7600MB for mouse or human genome (other species have a much smaller memory footprint), when performing read mapping. Using less memory will increase read mapping time.
-o $< basename >$ (`basename`)	Specify the base name of the index to be created.
-v	Output version of the program.

4.4 Read mapping

The Subread aligner (`subread-align` program in SourceForge Subread package or `align` function in Bioconductor Rsubread package) extracts a number of subreads from each read and then uses these subreads to vote for the mapping location of the read. It uses the the "seed-and-vote" paradigm for read mapping and reports the largest mappable region for each read. Table 2 describes the arguments used by Subread aligner (and also `Subjunc` aligner). Arguments used in Bioconductor `Rsubread` package are included in parenthesis.

Table 2: Arguments used by the `subread-align`/`subjunc` programs included in the SourceForge Subread package. Arguments in parenthesis in the first column are the equivalent arguments used in Bioconductor Rsubread package. Arguments used by `subread-align` only are marked with *. Arguments used by `subjunc` only are marked with **.

Arguments	Description
-b (`color2base=TRUE`)	Output base-space reads instead of color-space reads in mapping output for color space data (eg. LifTech SOLiD data). Note that the mapping itself will still be performed at color-space.
-B < *int* > (`nBestLocations`)	Specify the maximal number of equally-best mapping locations allowed to be reported for each read. 1 by default. 'NH' tag is used to indicate how many alignments are reported for the read and 'HI' tag is used for numbering the alignments reported for the same read, in the output. Note that `-u` option takes precedence over `-B`.
-d < *int* > (`minFragLength`)	Specify the minimum fragment/template length, 50 by default. Note that if the two reads from the same pair do not satisfy the fragment length criteria, they will be mapped individually as if they were single-end reads.
-D < *int* > (`maxFragLength`)	Specify the maximum fragment/template length, 600 by default.
-i < *index* > (`index`)	Specify the base name of the index.
-I < *int* > (`indels`)	Specify the number of INDEL bases allowed in the mapping. 5 by default. Indels of up to 200bp long can be detected.
-m < *int* > (`TH1`)	Specify the consensus threshold, which is the minimal number of consensus subreads required for reporting a hit. The consensus subreads are those subreads which vote for the same location in the reference genome for the read. If pair-end read data are provided, at least one of the two reads from the same pair must satisfy this criteria. 3 by default.
-M < *int* > (`maxMismatches`)	Specify the maximum number of mis-matched bases allowed in the alignment. 3 by default. Mis-matches found in soft-clipped bases are not counted.

-n < *int* > (`nsubreads`)	Specify the number of subreads extracted from each read, 10 by default.
-o < *output* > (`output_file`)	Give the name of output file. The default output format is BAM. All reads are included in mapping output, including both mapped and unmapped reads, and they are in the same order as in the input file.
-p < *int* > (`TH2`)	Specify the minimum number of consensus subreads both reads from the same pair must have. This argument is only applicable for paired-end read data. The value of this argument should not be greater than that of '-m' option, so as to rescue those read pairs in which one read has a high mapping quality but the other does not. 1 by default.
-P < 3 : 6 > (`phredOffset`)	Specify the format of Phred scores used in the input data, '3' for phred+33 and '6' for phred+64. '3' by default. For `align` function in Rsubread, the possible values are '33' (for phred+33) and '64' (for phred+64). '33' by default.
-r < *input* > (`readfile1`)	Give the name of input file(s) (multiple files are allowed to be provided to `align` and `subjunc` functions in Rsubread). For paired-end read data, this gives the first read file and the other read file should be provided via the -R option. Supported input formats include FASTQ/FASTA (uncompressed or gzip compressed)(default), SAM and BAM.
-R < *input* > (`readfile2`)	Provide name of the second read file from paired-end data. The program will switch to paired-end read mapping mode if this file is provided. (multiple files are allowed to be provided to `align` and `subjunc` functions in Rsubread).
-S < $ff : fr : rf$ > (`PE_orientation`)	Specify the orientation of the two reads from the same pair. It has three possible values including 'fr', 'ff' and ''rf. Letter 'f' denotes the forward strand and letter 'r' the reverse strand. 'fr' by default (ie. the first read in the pair is on the forward strand and the second read on the reverse strand).
* -t < *int* > (`type`)	Specify the type of input sequencing data. Possible values include `0`, denoting RNA-seq data, or `1`, denoting genomic DNA-seq data. Character values including 'rna' and 'dna' can also be used in the R function. For genomic DNA-seq data, the aligner takes into account both the number of matched bases and the number of mis-matched bases to determine the the best mapping location after applying the 'seed-and-vote' approach for read mapping. For RNA-seq data, only the number of mis-matched bases is considered for determining the best mapping location.
-T < *int* > (`nthreads`)	Specify the number of threads/CPUs used for mapping. The value should be between 1 and 32. 1 by default.

-u (`unique=TRUE`)	Output uniquely mapped reads only. Reads that were found to have more than one best mapping location will not be reported.
**−−allJunctions (`reportAllJunctions=TRUE`)	This option should be used with `subjunc` for detecting canonical exon-exon junctions (with 'GT/AG' donor/receptor sites), non-canonical exon-exon junctions and structural variants (SVs) in RNA-seq data. detected junctions will be saved to a file with suffix name ".junction.bed". Detected SV breakpoints will be saved to a file with suffix name ".breakpoints.txt", which includes chromosomal coordinates of detected SV breakpoints and also number of supporting reads. In the read mapping output, each breakpoint-containing read will contain the following extra fields for the description of its secondary alignment: CC(Chr), CP(Position),CG(CIGAR) and CT(strand). The primary alignment (described in the main field) and secondary alignment give respectively the mapping results for the two segments from the same read that were seperated by the breakpoint. Note that each breakpoint-containing read occupies only one row in mapping output. The mapping output includes mapping results for all the reads.
−−BAMinput (`input_format="BAM"`)	Specify that the input read data are in BAM format.
−−complexIndels	Detect multiple short indels that occur concurrently in a small genomic region (these indels could be as close as 1bp apart).
−−DPGapExt $< int >$ (`DP_GapExtPenalty`)	Specify the penalty for extending the gap when performing the Smith-Waterman dynamic programming. 0 by defaut.
−−DPGapOpen $< int >$ (`DP_GapOpenPenalty`)	Specify the penalty for opening a gap when applying the Smith-Waterman dynamic programming to detecting indels. -2 by defaut.
−−DPMismatch $< int >$ (`DP_MismatchPenalty`)	Specify the penalty for mismatches when performing the Smith-Waterman dynamic programming. 0 by defaut.
−−DPMatch $< int >$ (`DP_MatchScore`)	Specify the score for the matched base when performing the Smith-Waterman dynamic programming. 2 by defaut.
−−rg $< string >$ (`readGroup`)	Add a $< tag : value >$ to the read group (RG) header in the mapping output.
−−rg-id $< string >$ (`readGroupID`)	Specify the read group ID. If specified, the read group ID will be added to the read group header field and also to each read in the mapping output.
−−SAMinput (`input_format="SAM"`)	Specify that the input read data are in SAM format.
−−SAMoutput (`output_format="SAM"`)	Specify that mapping results are saved into a SAM format file.

*−−sv (`detectSV=TRUE`)	This option should be used with `subread-align` for detecting structural variants (SVs) in genomic DNA sequencing data. Detected SV breakpoints will be saved to a file with suffix name ".breakpoints.txt", which includes chromosomal coordinates of detected SV breakpoints and also number of supporting reads for each SV event. In the read mapping output, each breakpoint-containing read will contain the following extra fields for the description of its secondary alignment: CC(Chr), CP(Position),CG(CIGAR) and CT(strand). The primary alignment (described in the main field) and secondary alignment give respectively the mapping results for the two segments from the same read that were seperated by the breakpoint. Note that each breakpoint-containing read occupies only one row in mapping output. The mapping output includes mapping results for all the reads.
−−trim5 < *int* > (`nTrim5`)	Trim off < *int* > number of bases from 5' end of each read. 0 by default.
−−trim3 < *int* > (`nTrim3`)	Trim off < *int* > number of bases from 3' end of each read. 0 by default.
-v	Output version of the program.

4.5 Mapping quality scores

Both Subread and Subjunc aligners output a mapping quality score (MQS) for each mapped read, computed as

$$MQS = \begin{cases} (\sum_{i \in b_m}(1-p_i) - \sum_{i \in b_{mm}}(1-p_i)) \times 60/L & \text{if uniquely mapped} \\ & \text{[MQS is reset to 0 if less than 0]} \\ 0 & \text{if mapped to } > 1 \text{ best location} \end{cases}$$

where L is the read length, p_i is the base-calling p-value for the ith base in the read, b_m is the set of locations of matched bases, and b_{mm} is the set of locations of mismatched bases.

Base-calling p values can be readily computed from the base quality scores. Read bases of high sequencing quality have low base-calling p values. Read bases that were found to be insertions are treated as matched bases in the MQS calculation. The MQS is a read-length normalized value and it is in the range [0, 60).

4.6 Mapping output

Read mapping results for each library will be saved to a BAM or SAM format file. Short indels detected from the read data will be saved to a text file ('.indel'). If '–sv' is specified when running `subread-align`, breakpoints detected from structural variant events will be output to a text file for each library as well ('.breakpoints.txt').

Chapter 5

Mapping reads generated by RNA sequencing technologies

5.1 A quick start for using SourceForge Subread package

An index must be built for the reference first and then the read mapping and/or junction detection can be carried out.

Step 1: Building an index

The following command can be used to build a base-space index. You can provide a list of FASTA files or a single FASTA file including all the reference sequences.

```
subread-buildindex -o my_index chr1.fa chr2.fa ...
```

For more details about index building, see Section 4.3.

Step 2: Aligning the reads

Subread

If the purpose of an RNA-seq experiment is to quantify gene-level expression and discover differentially expressed genes, the Subread aligner is recommended. Subread carries out local alignments for RNA-seq reads. The commands used by Subread to align RNA-seq reads are the same as those used to align gDNA-seq reads. Below is an example of using Subread to map single-end RNA-seq reads.

```
subread-align -i my_index -r rnaseq-reads.txt -o subread_results.sam
```

Another RNA-seq aligner included in this package is the Subjunc aligner. Subjunc not only performs read alignments but also detects exon-exon junctions. The main difference between

Subread and Subjunc is that Subread does not attempt to detect exon-exon junctions in the RNA-seq reads. For the alignments of the exon-spanning reads, Subread just uses the largest mappable regions in the reads to find their mapping locations. This makes Subread more computationally efficient. The largest mappable regions can then be used to reliably assign the reads to their target genes by using a read summarization program (eg. featureCounts, see Section 6.2), and differential expression analysis can be readily performed based on the read counts yielded from read summarization. Therefore, Subread is sufficient for read mapping if the purpose of RNA-seq analysis is to perform a differential expression analysis. Also, Subread could report more mapped reads than Subjunc. For example, the exon-spanning reads that are not aligned by Subjunc due to the lack of canonical GT/AG splicing signals can be aligned by Subread as long as they have a good match with the reference sequence.

Subjunc

For other purposes of the RNA-seq data anlayses such as exon-exon junction detection, alternative splicing analysis and genomic mutation detection, Subjunc aligner should be used because exon-spanning reads need to be fully aligned. Below is an example command of using Subjunc to perform global alignments for paired-end RNA-seq reads. Note that there are two files produced after mapping: one is a BAM-format file including mapping results and the other a BED-format file including discovered exon-exon junctions.

```
subjunc -i my_index -r rnaseq-reads1.txt -R rnaseq-reads2.txt -o subjunc_result
```

5.2 A quick start for using Bioconductor **Rsubread** package

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Building an index

To build the index, you must provide a single FASTA file (eg. "genome.fa") which includes all the reference sequences.

```
library(Rsubread)
buildindex(basename="my_index",reference="genome.fa")
```

Step 2: Aligning the reads

Please refer to Section 5.1 for difference between Subread and Subjunc in mapping RNA-seq data. Below is an example for mapping a single-end RNA-seq dataset using Subread. Useful information about `align` function can be found in its help page (type `?align` in your R prompt).

```
align(index="my_index",readfile1="rnaseq-reads.txt.gz",output_file="subread_results.bam")
```

Below is an example for mapping a single-end RNA-seq dataset using Subjunc. Useful information about `subjunc` function can be found in its help page (type `?subjunc` in your R prompt).

```
subjunc(index="my_index",readfile1="rnaseq-reads.txt.gz",output_file="subjunc_results.bam")
```

5.3 Local read alignment

The `Subread` and `Subjunc` can both be used to map RNA-seq reads to the reference genome. If the goal of the RNA-seq data is to perform expression analysis, eg. finding genes expressing differentially between different conditions, then `Subread` is recommended. `Subread` performs fast local alignments for reads and reports the mapping locations that have the largest overlap with the reads. These reads can then be assigned to genes for expression analysis. For this type of analysis, global alignments for the exon-spanning reads are not required because local aligments are sufficient to get reads to be accurately assigned to genes.

However, for other types of RNA-seq data analyses such as exon-exon junction discovery, genomic mutation detection and allele-specific gene expression analysis, global alignments are required. The next section describes the Subjunc aligner, which performs global aligments for RNA-seq reads.

5.4 Global read alignment

Subjunc aligns each exon-spanning read by firstly using a large number of subreads extracted from the read to identify multiple target regions matching the selected subreads, and then using the splicing signals (donor and receptor sites) to precisely determine the mapping locations of the read bases. It also includes a verification step to compare the quality of mapping reads as exon-spanning reads with the quality of mapping reads as exonic reads to finally decide how to best map the reads. Reads may be re-aligned if required.

Output of Subjunc aligner includes a list of discovered exon-exon junction locations and also the complete alignment results for the reads. Table 2 describes the arguments used by the Subjunc program.

5.5 Mapping output

Read mapping results for each library will be saved to a BAM/SAM file. Detected exon-exon junctions will be saved to a BED file for each library ('.junction.bed'). Detected short indels will be saved to a text file ('.indel').

5.6 Mapping microRNA sequencing reads (miRNA-seq)

To use Subread aligner to map miRNA-seq reads, a full index must be built for the reference genome before read mapping can be carried out. For example, the following command builds a full index for mouse reference genome *mm10*:

```
subread-buildindex -F -B -o mm10_full_index mm10.fa
```

The full index includes 16bp mers extracted from every genomic location in the genome. Note that if `-F` is not specified, `subread-buildindex` builds a gapped index which includes 16bp mers extracted every three bases in the reference genome, ie. there is a 2bp gap between each pair of neighbouring 16bp mers.

After the full index was built, read alignment can be performed. Reads do not need to be trimmed before feeding them to Subread aligner since Subread soft clips sequences in the reads that can not be properly mapped. The parameters used for mapping miRNA-seq reads need to be carefully designed due to the very short length of miRNA sequences (∼22bp). The total number of subreads (16bp mers) extracted from each read should be the read length minus 15, which is the maximum number of subreads that can be possibly extracted from a read. The reason why we need to extract the maximum number of subreads is to achieve a high sensitivity in detecting the short miRNA sequences.

The threshold for the number of consensus subreads required for reporting a hit should be within the range of 2 to 7 consensus subreads inclusive. The larger the number of consensus subreads required, the more stringent the mapping will be. Using a threshold of 2 consensus subreads allows the detection of miRNA sequences of as short as 17bp, but the mapping error rate could be relatively high. With this threshold, there will be at least 17 perfectly matched bases present in each reported alignment. If a threshold of 4 consensus subreads was used, length of miRNA sequences that can be detected is 19 bp or longer. With this threshold, there will be at least 19 perfectly matched bases present in each reported alignment. When a threshold of 7 consensus subreads was used, only miRNA sequences of 22bp or longer can be detected (at least 22 perfectly matched bases will be present in each reported alignment).

We found that there was a significant decrease in the number of mapped reads when the requried number of consensus subreads increased from 4 to 5 when we tried to align a mouse miRNA-seq dataset, suggesting that there are a lot of miRNA sequences that are only 19bp long. We therefore used a threshold of 4 consensus subreads to map this dataset. However, what we observed might not be the case for other datasets that were generated from different cell types and different species.

Below is an example of mapping 50bp long reads (adaptor sequences were included in the reads in addition to the miRNA sequences), with at least 4 consensus subreads required in the mapping:

```
subread-align -i mm10_full_index -n 35 -m 4 -M 3 -T 10 -I 0 -P 3 -B 10
-r miRNA_reads.fastq -o result.sam
```

The '-B 10' parameter instructs Subread aligner to report up to 10 best mapping locations (equally best) in the mapping results. The multiple locations reported for the reads could be useful for investigating their true origin, but they might need to be filtered out when assigning mapped reads to known miRNA genes to ensure a high-quality quantification of miRNA genes. The miRBase database (`http://www.mirbase.org/`) is a useful resource that includes annotations for miRNA genes in many species. The featureCounts program can be readily used for summarizing reads to miRNA genes.

Chapter 6

Read summarization

6.1 Introduction

Sequencing reads often need to be assigned to genomic features of interest after they are mapped to the reference genome. This process is often called *read summarization* or *read quantification*. Read summarization is required by a number of downstream analyses such as gene expression analysis and histone modification analysis. The output of read summarization is a count table, in which the number of reads assigned to each feature in each library is recorded.

A particular challenge to the read summarization is how to deal with those reads that overlap more than one feature (eg. an exon) or meta-feature (eg. a gene). Care must be taken to ensure that such reads are not over-counted or under-counted. Here we describe the featureCounts program, an efficient and accurate read quantifier. featureCounts has the following features:

- It carries out precise and accurate read assignments by taking care of indels, junctions and structural variants in the reads.
- It takes only ∼1 minute to summarize 20 million read pairs of reads to 26 thousand RefSeq genes.
- It supports GTF/SAF format annotation and SAM/BAM read data.
- It supports strand-specific read summarization.
- It can perform read summarization at both feature level (eg. exon level) and meta-feature level (eg. gene level).
- It allows users to specify whether reads overlapping with more than one feature should be counted or not.
- It gives users full control on the summarization of paired-end reads, including allowing them to check if both ends are mapped and/or if the fragment length falls within the specified range.

- It can discriminate the features that were overlapped by both ends of the fragment from the features that were overlapped by only one end of the same fragment to get more accurate read assignments.
- It allows users to specify whether chimeric fragments should be counted.
- It automatically detects the read input format (SAM or BAM).
- It automatically re-order paired-end reads if reads belonging to the same pair are not adjacent to each other in input read files.

6.2 featureCounts

6.2.1 Input data

The data input to featureCounts consists of (i) one or more files of aligned reads in either SAM or BAM format and (ii) a list of genomic features in either Gene Transfer Format (GTF) or General Feature Format (GFF) or Simplified Annotation Format (SAF). The format of input reads is automatically detected (SAM or BAM).

For paired-end reads, if they were location-sorted in the input featureCounts will automatically re-order the reads to place next to each other the reads from the same pair before counting them. We also provide an utility program repair to allow users to pair up the reads before feeding them to featureCounts. Note that name-sorted paired-end reads generated by other programs may include incorrectly paired reads due to for example multi-mapping issue. If this is the case, featureCounts will re-sort them.

Both read alignment and read counting should use the same reference genome. For each read, the BAM/SAM file gives the name of the reference chromosome or contig the read mapped to, the start position of the read on the chromosome or contig/scaffold, and the so-called CIGAR string giving the detailed alignment information including insertions and deletions and so on relative to the start position.

The genomic features can be specified in either GTF/GFF or SAF format. The SAF format is the simpler and includes only five required columns for each feature (see next section). In either format, the feature identifiers are assumed to be unique, in accordance with commonly used Gene Transfer Format (GTF) refinement of GFF.

featureCounts supports strand-specific read counting if strand-specific information is provided. Read mapping results usually include mapping quality scores for mapped reads. Users can optionally specify a minimum mapping quality score that the assigned reads must satisfy.

6.2.2 Annotation format

The genomic features can be specified in either GTF/GFF or SAF format. A definition of the GTF format can be found at UCSC website (`http://genome.ucsc.edu/FAQ/FAQformat.html#format4`). The SAF format includes five required columns for each feature: feature identifier, chromosome name, start position, end position and strand. These five columns

provide the minimal sufficient information for read quantification purposes. Extra annotation data are allowed to be added from the sixth column.

A SAF-format annotation file should be a tab-delimited text file. It should also include a header line. An example of a SAF annotation is shown as below:

```
GeneID Chr Start End Strand
497097 chr1 3204563 3207049 -
497097 chr1 3411783 3411982 -
497097 chr1 3660633 3661579 -
100503874 chr1 3637390 3640590 -
100503874 chr1 3648928 3648985 -
100038431 chr1 3670236 3671869 -
...
```

`GeneID` column includes gene identifiers that can be numbers or character strings. Chromosomal names included in the `Chr` column must match the chromosomal names of reference sequences to which the reads were aligned.

6.2.3 Single and paired-end reads

Reads may be paired or unpaired. If paired reads are used, then each pair of reads defines a DNA or RNA fragment bookended by the two reads. In this case, featureCounts can be instructed to count fragments rather than reads. featureCounts automatically sorts reads by name if paired reads are not in consecutive positions in the SAM or BAM file. Users do not need sort their paired reads before providing them to featureCounts.

6.2.4 Features and meta-features

featureCounts is a general-purpose read summarization function, which assigns mapped reads (RNA-seq reads or genomic DNA-seq reads) to genomic features or meta-features. Each feature is an interval (range of positions) on one of the reference sequences. We define a meta-feature to be a set of features representing a biological construct of interest. For example, features often correspond to exons and meta-features to genes. Features sharing the same feature identifier in the GTF or SAF annotation are taken to belong to the same meta-feature. featureCounts can summarize reads at either the feature or meta-feature levels.

We recommend to use unique gene identifiers, such as NCBI Entrez gene identifiers, to cluster features into meta-features. Gene names are not recommended to use for this purpose because different genes may have the same names. Unique gene identifiers were often included in many publicly available GTF annotations which can be readily used for summarization. The Bioconductor Rsubread package also includes NCBI RefSeq annotations for human and mice. Entrez gene identifiers are used in these annotations.

6.2.5 Overlap of reads with features

featureCounts preforms precise read assignment by comparing mapping location of every base in the read or fragment with the genomic region spanned by each feature. It takes account of any gaps (insertions, deletions, exon-exon junctions or structural variants) that are found in the read. It calls a hit if any overlap (1bp or more) is found between the read or fragment and a feature. A hit is called for a meta-feature if the read or fragment overlaps any component feature of the meta-feature.

6.2.6 Multiple overlaps

A multi-overlap read or fragment is one that overlaps more than one feature, or more than one meta-feature when summarizing at the meta-feature level. featureCounts provides users with the option to either exclude multi-overlap reads or to count them for each feature that is overlapped. The decision whether or not to counting these reads is often determined by the experiment type. We recommend that reads or fragments overlapping more than one gene are not counted for RNA-seq experiments, because any single fragment must originate from only one of the target genes but the identity of the true target gene cannot be confidently determined. On the other hand, we recommend that multi-overlap reads or fragments are counted for most ChIP-seq experiments because epigenetic modifications inferred from these reads may regulate the biological functions of all their overlapping genes.

Note that, when counting at the meta-feature level, reads that overlap multiple features of the same meta-feature are always counted exactly once for that meta-feature, provided there is no overlap with any other meta-feature. For example, an exon-spanning read will be counted only once for the corresponding gene even if it overlaps with more than one exon.

6.2.7 In-built annotations

In-built gene annotations for genomes *hg19*, *mm10* and *mm9* are included in both Bioconductor Rsubread package and SourceForge Subread package. These annotations were downloaded from NCBI RefSeq database and then adapted by merging overlapping exons from the same gene to form a set of disjoint exons for each gene. Genes with the same Entrez gene identifiers were also merged into one gene.

Each row in the annotation represents an exon of a gene. There are five columns in the annotation data including Entrez gene identifier (*GeneID*), chromosomal name (*Chr*), chromosomal start position(*Start*), chromosomal end position (*End*) and strand (*Strand*).

In Rsubread, users can access these annotations via the getInBuiltAnnotation function. In Subread, these annotations are stored in directory 'annotation' under home directory of the package.

6.2.8 Program output

Output of featureCounts program in SourceForge Subread package is saved into a tab-delimited file, which includes annotation columns ('Geneid', 'Chr', 'Start', 'End', 'Strand' and 'Length')

and data columns (read counts for each gene in each library). Annotation column 'Length' contains total number of non-overlapping bases of each feature or meta-feature. When for example summarizing RNA-seq reads to genes, this column will give total number of non-overlapping bases included in all exons belonging to the same gene, for each gene.

When performing summarization at meta-feature level, annotation columns including 'Chr', 'Start', 'End', 'Strand' and 'Length' give the annotation information for every feature included each meta-features. Therefore, each of these columns may include more than one value (semi-colon separated).

Output of featureCounts program in SourceForge Subread package also includes stat info of summarization results, which is saved to a tab-delimited file as well (a separate file). This file gives the total number of reads that are successfully assigned and also numbers of reads that are not assigned due to various reasons. Below lists the reasons why reads may not be assigned:

- Unassigned_Ambiguity: overlapping with two or more features (feature-level summarization) or meta-features (meta-feature-level) summarization.
- Unassigned_MultiMapping: reads marked as multi-mapping in SAM/BAM input (the 'NH' tag is checked by the program).
- Unassigned_NoFeatures: not overlapping with any features included in the annotation.
- Unassigned_Unmapped: reads are reported as unmapped in SAM/BAM input. Note that if the '–primary' option of featureCounts program is specified, the read marked as a primary alignment will be considered for assigning to features.
- Unassigned_MappingQuality: mapping quality scores lower than the specified threshold.
- Unassigned_FragementLength: length of fragment does not satisfy the criteria.
- Unassigned_Chimera: two reads from the same pair are mapped to different chromosomes or have incorrect orientation.
- Unassigned_Secondary: reads marked as second alignment in the FLAG field in SAM/BAM input.
- Unassigned_Nonjunction: reads do not span two or more exons. Such reads will not be assigned if the '–countSplitAlignmentsOnly' option is specified.
- Unassigned_Duplicate: reads marked as duplicate in the FLAG field in SAM/BAM input.

All these output were also provided by the featureCounts function included in Bioconductor Rsubread package, except that read summarization results are saved into an R 'List' object. For more details, see the help page for featureCounts function in Rsubread.

6.2.9 Program usage

Table 3 describes the parameters used by the featureCounts program.

Table 3: arguments used by the featureCounts program included in the SourceForge Subread package. Arguments included in parenthesis are the equivalent parameters used by featureCounts function in Bioconductor Rsubread package.

Arguments	Description
input_files (`files`)	Give the names of input read files that include the read mapping results. The program automatically detects the file format (SAM or BAM). Multiple files can be provided at the same time.
-a < *input* > (`annot.ext, annot.inbuilt`)	Give the name of an annotation file.
-A (`chrAliases`)	Give the name of a file that contains aliases of chromosome names. The file should be a comma delimited text file that includes two columns. The first column gives the chromosome names used in the annotation and the second column gives the chromosome names used by reads. This file should not contain header lines. Names included in this file are case sensitive.
-B (`requireBothEndsMapped`)	If specified, only fragments that have both ends successfully aligned will be considered for summarization. This option should be used together with `-p` (or `isPairedEnd` in Rsubread featureCounts).
-C (`countChimericFragments`)	If specified, the chimeric fragments (those fragments that have their two ends aligned to different chromosomes) will NOT be counted. This option should be used together with `-p` (or `isPairedEnd` in Rsubread featureCounts).
-d < *int* > (`minFragLength`)	Minimum fragment/template length, 50 by default. This option must be used together with `-p` and `-P`.
-D < *int* > (`maxFragLength`)	Maximum fragment/template length, 600 by default. This option must be used together with `-p` and `-P`.
-f (`useMetaFeatures`)	If specified, read summarization will be performed at feature level (eg. exon level). Otherwise, it is performed at meta-feature level (eg. gene level).
-F (`isGTFAnnotationFile`)	Specify the format of the annotation file. Acceptable formats include 'GTF' and 'SAF' (see Section 6.2.2 for details). The C version of featureCounts program uses a GTF format annotation by default, but the R version uses a SAF format annotation by default. The R version also includes in-built annotations.
-g < *input* > (`GTF.attrType`)	Specify the attribute type used to group features (eg. exons) into meta-features (eg. genes) when GTF annotation is provided. 'gene_id' by default. This attribute type is usually the gene identifier. This argument is useful for the meta-feature level summarization.

-G < *input* > (`genome`)	Provide the name of a FASTA-format file that includes the reference genome sequences. The reference genome provided here should be the same as the one used in read mapping.
-J (`juncCounts`)	Count the number of reads supporting each exon-exon junction. Junctions are identified from those exon-spanning reads (containing 'N' in CIGAR string) in input data. For each junction, the reported data include number of supporting reads, genes that the junction belongs to, chromosomal coordinates of splice sites etc.
-M (`countMultiMappingReads`)	If specified, multi-mapping reads/fragments will be counted. A multi-mapping read will be counted up to N times if it has N reported mapping locations. The program uses the 'NH' tag to find multi-mapping reads.
-o < *input* >	Give the name of the output file. The output file contains the number of reads assigned to each meta-feature (or each feature if `-f` is specified). Note that the featureCounts function in Rsubread does not use this parameter. It returns a `list` object including read summarization results and other data.
-O (`allowMultiOverlap`)	If specified, reads (or fragments if `-p` is specified) will be allowed to be assigned to more than one matched meta-feature (or feature if `-f` is specified). Reads/fragments overlapping with more than one meta-feature/feature will be counted more than once. Note that when performing meta-feature level summarization, a read (or fragment) will still be counted once if it overlaps with multiple features belonging to the same meta-feature but does not overlap with other meta-features.
-p (`isPairedEnd`)	If specified, fragments (or templates) will be counted instead of reads. This option is only applicable for paired-end reads.
-P (`checkFragLength`)	If specified, the fragment length will be checked when assigning fragments to meta-features or features. This option must be used together with `-p`. The fragment length thresholds should be specified using `-d` and `-D` options.
-Q < *int* > (`minMQS`)	The minimum mapping quality score a read must satisfy in order to be counted. For paired-end reads, at least one end should satisfy this criteria. 0 by default.
-R	Output read assignment results for each read (or fragment if paired end). They are saved to a tab-delimited file that contains four columns including read name, status(assigned or the reason if not assigned), name of target feature/meta-feature and number of hits if the read/fragment is counted multiple times. Name of the file is the input file name added with a suffix '.featureCounts'.

-s $< int >$ (`isStrandSpecific`)	Indicate if strand-specific read counting should be performed. It has three possible values: 0 (unstranded), 1 (stranded) and 2 (reversely stranded). 0 by default. For paired-end reads, strand of the first read is taken as the strand of the whole fragment and FLAG field of the current read is used to tell if it is the first read in the fragment.
-S $< ff : fr : rf >$ (`PE_orientation`)	Specify the orientation of the two reads from the same pair. It has three possible values including 'fr', 'ff' and ''rf. Letter 'f' denotes the forward strand and letter 'r' the reverse strand. 'fr' by default (ie. the first read in the pair is on the forward strand and the second read on the reverse strand).
-t $< input >$ (`GTF.featureType`)	Specify the feature type. Only rows which have the matched feature type in the provided GTF annotation file will be included for read counting. 'exon' by default.
-T $< int >$ (`nthreads`)	Number of the threads. The value should be between 1 and 32. 1 by default.
-v	Output version of the program.
−−countSplitAlignmentsOnly (`splitOnly`)	If specified, only split alignments (CIGAR strings contain letter 'N') will be counted. All the other alignments will be ignored. An example of split alignments is the exon-spanning reads in RNA-seq data. If exon-spanning reads need to be assigned to all their overlapping exons, '-f' and '-O' options should be provided as well.
−−countNonSplitAlignmentsOnly (`nonSplitOnly`)	If specified, only non-split alignments (CIGAR strings do not contain letter 'N') will be counted. All the other alignments will be ignored.
−−donotsort (`autosort`)	If specified, paired end reads will not be re-ordered even if reads from the same pair were found not to be next to each other in the input.
−−fraction (`fraction`)	If specified, a fractional count 1/n will be generated for each multi-mapping read, where n is the number of alignments (indicated by 'NH' tag) reported for the read. This option must be used together with the '-M' option.
−−ignoreDup (`ignoreDup`)	If specified, reads that were marked as duplicates will be ignored. Bit Ox400 in FLAG field of SAM/BAM file is used for identifying duplicate reads. In paired end data, the entire read pair will be ignored if at least one end is found to be a duplicate read.
−−largestOverlap (`largestOverlap`)	If specified, reads (or fragments) will be assigned to the target that has the largest number of overlapping bases.

−−maxMOp < *int* > (`maxMOp`)	Specify the maximum number of 'M' operations (matches or mis-matches) allowed in a CIGAR string. 10 by default. Both 'X' and '=' operations are treated as 'M' and adjacent 'M' operations are merged in the CIGAR string. When the number of 'M' operations exceeds the limit, only the first 'maxMOp' number of 'M' operations will be used in read assignment.
−−minOverlap < *int* > (`minOverlap`)	Specify the minimum required number of overlapping bases between a read (or a fragment) and an overlapping feature. 1 by default. If a negative value is provided, the read will be extended from both ends.
−−primary (`primaryOnly`)	If specified, only primary alignments will be counted. Primary and secondary alignments are identified using bit 0x100 in the Flag field of SAM/BAM files. All primary alignments in a dataset will be counted no matter they are from multi-mapping reads or not (ie. '-M' is ignored).
−−read2pos < *int* > (`read2pos`)	The read is reduced to its 5' most base or 3' most base. Read summarization is then performed based on the single base position to which the read is reduced. By default, no read reduction will be performed.
−−readExtension5 < *int* > (`readExtension5`)	Reads are extended upstream by < *int* > bases from their 5' end. 0 by default.
−−readExtension3 < *int* > (`readExtension3`)	Reads are extended downstream by < *int* > bases from their 3' end. 0 by default.

6.3 A quick start for featureCounts in SourceForge Subread

You need to provide read mapping results (in either SAM or BAM format) and an annotation file for the read summarization. The example commands below assume your annotation file is in GTF format.

Summarize SAM format single-end reads using 5 threads:

```
featureCounts -T 5 -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_SE.sam
```

Summarize BAM format single-end read data:

```
featureCounts -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_SE.bam
```

Summarize multiple libraries at the same time:

```
featureCounts -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results1.bam mapping_results2.bam
```

Summarize paired-end reads and count fragments (instead of reads):

```
featureCounts -p -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.bam
```

Count fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:

```
featureCounts -p -P -d 50 -D 600 -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.bam
```

Count fragments which have both ends successfully aligned without considering the fragment length constraint:

```
featureCounts -p -B -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.bam
```

Exclude chimeric fragments from the fragment counting:

```
featureCounts -p -C -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.bam
```

6.4 A quick start for featureCounts in Bioconductor Rsubread

You need to provide read mapping results (in either SAM or BAM format) and an annotation file for the read summarization. The example commands below assume your annotation file is in GTF format.

Load Rsubread library from you R session:

```
library(Rsubread)
```

Summarize single-end reads using built-in RefSeq annotation for mouse genome mm9:

```
featureCounts(files="mapping_results_SE.sam",annot.inbuilt="mm9")
```

Summarize single-end reads using a user-provided GTF annotation file:

```
featureCounts(files="mapping_results_SE.sam",annot.ext="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

Summarize single-end reads using 5 threads:

```
featureCounts(files="mapping_results_SE.sam",nthreads=5)
```

Summarize BAM format single-end read data:

```
featureCounts(files="mapping_results_SE.bam")
```

Summarize multiple libraries at the same time:

```
featureCounts(files=c("mapping_results1.bam","mapping_results2.bam"))
```

Summarize paired-end reads and counting fragments (instead of reads):

```
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE)
```

Count fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:

```
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,checkFragLength=TRUE,
minFragLength=50,maxFragLength=600)
```

Count fragments which have both ends successfully aligned without considering the fragment length constraint:

```
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,requireBothEndsMapped=TRUE)
```

Exclude chimeric fragments from fragment counting:

```
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,countChimericFragments=FALSE)
```

Chapter 7

SNP calling

7.1 Algorithm

SNPs(Single Nucleotide Polymorphisms) are the mutations of single nucleotides in the genome. It has been reported that many diseases were initiated and/or driven by such mutations. Therefore, successful detection of SNPs is very useful in designing better diagnosis and treatments for a variety of diseases such as cancer. SNP detection also is an important subject of many population studies.

Next-gen sequencing technologies provide an unprecedented opportunity to identify SNPs at the highest resolution. However, it is extremely computing-intensive to analyze the data generated from these technologies for the purpose of SNP discovery because of the sheer volume of the data and the large number of chromosomal locations to be considered. To discover SNPs, reads need to be mapped to the reference genome first and then all the read data mapped to a particular site will be used for SNP calling for that site. Discovery of SNPs is often confounded by many sources of errors. Mapping errors and sequencing errors are often the major sources of errors causing incorrect SNP calling. Incorrect alignments of indels, exon-exon junctions and structural variants in the reads can also result in wrong placement of blocks of continuous read bases, likely giving rise to consecutive incorrectly reported SNPs.

We have developed a highly accurate and efficient SNP caller, called *exactSNP* [9]. *exactSNP* calls SNPs for individual samples, without requiring control samples to be provided. It tests the statistical significance of SNPs by comparing SNP signals to their background noises. It has been found to be an order of magnitude faster than existing SNP callers.

7.2 exactSNP

Below is the command for running `exactSNP` program. The complete list of parameters used by `exactSNP` can be found in Table 4.

```
exactSNP [options] -i input -g reference_genome -o output
```

Table 4: arguments used by the `exactSNP` program included in the SourceForge Subread package. Arguments included in parenthesis are the equivalent parameters used by `exactSNP` function in Bioconductor Rsubread package.

Arguments	Description
-a $< file >$ (SNPAnnotationFile)	Specify name of a VCF-format file that includes annotated SNPs. Such annotation files can be downloaded from public databases such as the dbSNP database. Incorporating known SNPs into SNP calling has been found to be helpful. However note that the annotated SNPs may or may not be called for the sample being analyzed.
-b (isBAM)	Indicate the input file provided via $-i$ is in BAM format.
-f $< float >$ (minAllelicFraction)	Specify the minimum fraction of mis-matched bases a SNP-containing location must have. Its value must between 0 and 1. 0 by default.
-g $< file >$ (refGenomeFile)	Specify name of the file including all reference sequences. Only one single FASTA format file should be provided.
-i $< file > [-b\ if\ BAM]$ $(readFile)$	Specify name of an input file including read mapping results. The format of input file can be SAM or BAM (`-b` needs to be specified if a BAM file is provided).
-n $< int >$ (minAllelicBases)	Specify the minimum number of mis-matched bases a SNP-containing location must have. 1 by default.
-o $< file >$ (outputFile)	Specify name of the output file. This program outputs a VCF format file that includes discovered SNPs.
-Q $< int >$ (qvalueCutoff)	Specify the q-value cutoff for SNP calling at sequencing depth of 50X. 12 by default. The corresponding p-value cutoff is 10^{-Q}. Note that this program automatically adjusts the q-value cutoff according to the sequencing depth at each chromosomal location.
-r $< int >$ (minReads)	Specify the minimum number of mapped reads a SNP-containing location must have (ie. the minimum coverage). 1 by default.
-s $< int >$ (minBaseQuality)	Specify the cutoff for base calling quality scores (Phred scores) read bases must satisfy to be used for SNP calling. 13 by default. Read bases that have Phred scores lower than the cutoff value will be excluded from the analysis.
-t $< int >$ (nTrimmedBases)	Specify the number of bases trimmed off from each end of the read. 3 by default.
-T $< int >$ (nthreads)	Specify the number of threads. 1 by default.
-v	Output version of the program.

-x < *int* > (maxReads)	Specify the maximum number of mapped reads a SNP-containing location could have. 3000 by default. Any location having more than the threshold number of reads will not be considered for SNP calling. This option is useful for removing PCR artefacts.

Chapter 8

Utility programs

Usage info for each utility program can be seen by just typing the program name on the command prompt.

8.1 repair

This program takes as input a paired-end BAM file and places reads from the same pair next to each other in its output. BAM files generated by repair are compatible with featureCounts program, ie they will not be re-sorted by featureCounts. Note that you do not have to run repair before running featureCounts. featureCounts calls repair automatically if it finds that reads need to be re-sorted.

The repair program uses a novel approach to quickly find reads from the same pair, rather than performing time-consuming sort of read names. It takes only about half a minute to re-order a location-sorted BAM file including 30 million read pairs.

8.2 coverageCount

Compute the read coverage for each chromosomal location in the genome.

8.3 propmapped

Get number of mapped reads from a BAM/SAM file.

8.4 qualityScores

Retrieve Phred scores for read bases from a Fastq/BAM/SAM file.

8.5 removeDup

Remove duplicated reads from a SAM file.

8.6 subread-fullscan

Get all chromosomal locations that contain a genomic sequence sharing high homology with a given input sequence.

Chapter 9

Case studies

9.1 A Bioconductor R pipeline for analyzing RNA-seq data

Here we illustrate how to use two Bioconductor packages - Rsubread and limma - to perform a complete RNA-seq analysis, including Subread read mapping, featureCounts read summarization, voom normalization and limma differential expresssion analysis.

Data and software. The RNA-seq data used in this case study include four libraries: A_1, A_2, B_1 and B_2. Sample A is Universal Human Reference RNA (UHRR) and sample B is Human Brain Reference RNA (HBRR). A_1 and A_2 are two replicates of sample A (undergoing separate sample preparation), and B_1 and B_2 are two replicates of sample B. In this case study, A_1 and A_2 are treated as biological replicates although they are more like technical replicates. B_1 and B_2 are treated as biological replicates as well.

Note that these libraries only included reads originating from human chromosome 1 (according to Subread aligner). Reads were generated by the MAQC/SEQC Consortium. Data used in this case study can be downloaded from
`http://bioinf.wehi.edu.au/RNAseqCaseStudy/data.tar.gz` (283MB). Both read data and reference sequence for chromosome 1 of human genome (GRCh37) were included in the data.

After downloading the data, you can uncompress it and save it to your current working directory. Launch R and load Rsubread, limma and edgeR libraries by issuing the following commands at your R prompt. Version of your R should be 3.0.2 or later. Rsubread version should be 1.12.1 or later and limma version should be 3.18.0 or later. Note that this case study only runs on Linux/Unix and Mac OS X.

```
library(Rsubread)
library(limma)
library(edgeR)
```

To install/update Rsubread and limma packages, issue the following commands at your R prompt:

```
source("http://bioconductor.org/biocLite.R")
biocLite(pkgs=c("Rsubread","limma","edgeR"))
```

Index building. Build an index for human chromosome 1. This typically takes ∼3 minutes. Index files with basename 'chr1' will be generated in your current working directory.

```
buildindex(basename="chr1",reference="hg19_chr1.fa")
```

Alignment. Perform read alignment for all four libraries and report uniquely mapped reads only. This typically takes ∼5 minutes. BAM files containing the mapping results will be generated in your current working directory.

```
targets <- readTargets()
align(index="chr1",readfile1=targets$InputFile,input_format="gzFASTQ",output_format="BAM",
output_file=targets$OutputFile,unique=TRUE,indels=5)
```

Read summarization. Summarize mapped reads to NCBI RefSeq genes. This will only take a few seconds. Note that the featureCounts function contains built-in RefSeq annotations for human and mouse genes. featureCounts returns an R 'List' object, which includes raw read count for each gene in each library and also annotation information such as gene identifiers and gene lengths.

```
fc <- featureCounts(files=targets$OutputFile,annot.inbuilt="hg19")

fc$counts[1:5,]
          A_1.bam A_2.bam B_1.bam B_2.bam
653635        642     522     591     596
100422834       1       0       0       0
645520          5       3       0       0
79501           0       0       0       0
729737         82      72      30      25

fc$annotation[1:5,c("GeneID","Length")]
     GeneID Length
1    653635   1769
2 100422834    138
3    645520   1130
4     79501    918
5    729737   3402
```

Create a DGEList object.

```
x <- DGEList(counts=fc$counts, genes=fc$annotation[,c("GeneID","Length")])
```

Calculate RPKM (reads per kilobases of exon per million reads mapped) values for genes:

```
x_rpkm <- rpkm(x,x$genes$Length,prior.count=0)

x_rpkm[1:5,]
          A_1.bam A_2.bam B_1.bam B_2.bam
653635        939   905.0     709     736
```

```
100422834      19     0.0       0       0
645520         11     8.1       0       0
79501           0     0.0       0       0
729737         62    64.9      19      16
```

Filtering. Only keep in the analysis those genes which had >10 reads per million mapped reads in at least two libraries.

```
isexpr <- rowSums(cpm(x) > 10) >= 2
x <- x[isexpr,]
```

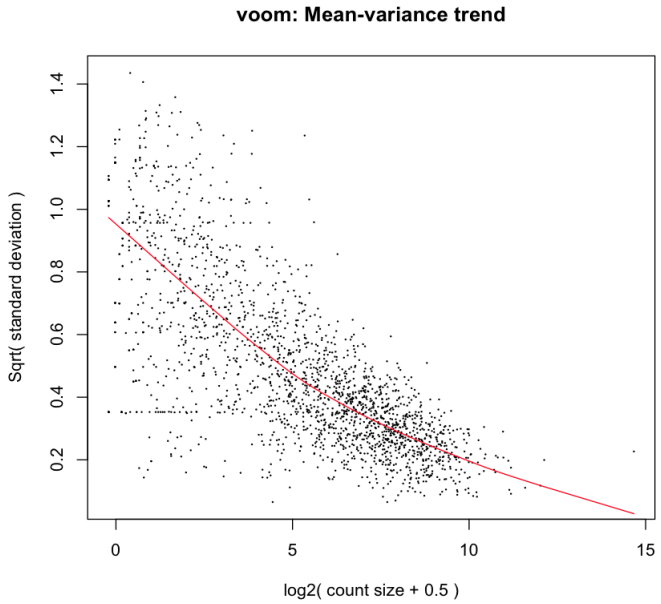
Design matrix. Create a design matrix:

```
celltype <- factor(targets$CellType)
design <- model.matrix(~0+celltype)
colnames(design) <- levels(celltype)
```

Normalization. Perform voom normalization:

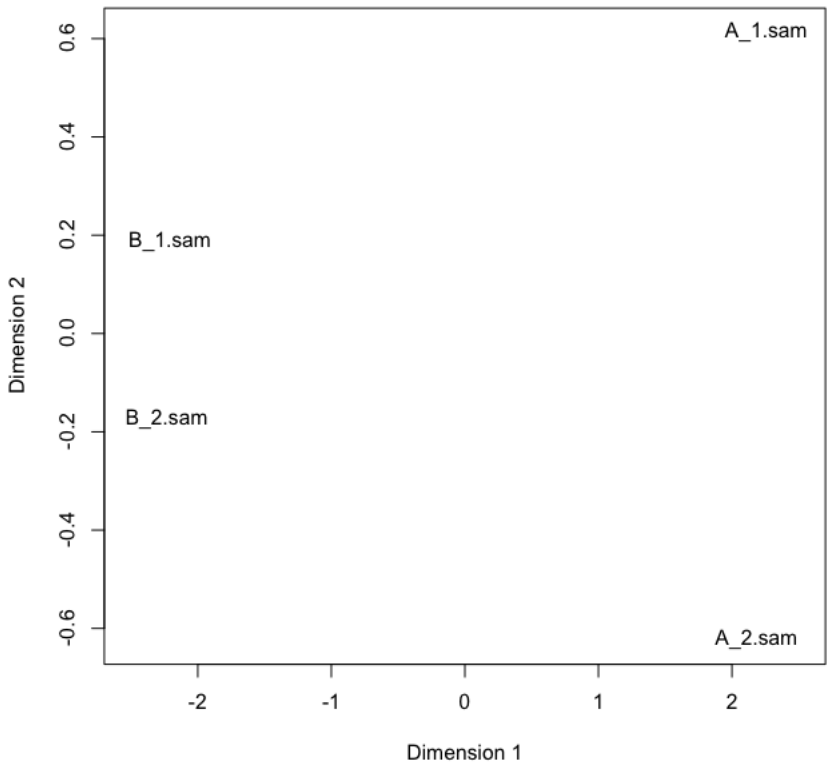
```
y <- voom(x,design,plot=TRUE)
```

The figure below shows the mean-variance relationship estimated by voom.



Sample clustering. Multi-dimensional scaling (MDS) plot shows that sample A libraries are clearly separated from sample B libraries.

```
plotMDS(y,xlim=c(-2.5,2.5))
```



Linear model fitting and differential expression analysis. Fit linear models to genes and assess differential expression using eBayes moderated t statistic. Here we compare sample B vs sample A.

```
fit <- lmFit(y,design)
contr <- makeContrasts(BvsA=B-A,levels=design)
fit.contr <- eBayes(contrasts.fit(fit,contr))
dt <- decideTests(fit.contr)
summary(dt)
   BvsA
-1  922
0   333
1   537
```

List top 10 differentially expressed genes:

```
options(digits=2)
topTable(fit.contr)
             GeneID Length logFC AveExpr   t P.Value adj.P.Val  B
100131754 100131754   1019   1.6      16 113 3.5e-28   6.3e-25 54
2023           2023   1812  -2.7      13 -91 2.2e-26   1.9e-23 51
2752           2752   4950   2.4      13  82 1.5e-25   9.1e-23 49
22883         22883   5192   2.3      12  64 1.8e-23   7.9e-21 44
6135           6135    609  -2.2      12 -62 3.1e-23   9.5e-21 44
6202           6202    705  -2.4      12 -62 3.2e-23   9.5e-21 44
```

4904	4904	1546	-3.0	11	-60	5.5e-23	1.4e-20	43
23154	23154	3705	3.7	11	55	2.9e-22	6.6e-20	41
8682	8682	2469	2.6	12	49	2.2e-21	4.3e-19	39
6125	6125	1031	-2.0	12	-48	3.1e-21	5.6e-19	39

Bibliography

[1] Y. Liao, G. K. Smyth, and W. Shi. The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41:e108, 2013.

[2] K. W. Tang, B. Alaei-Mahabadi, T. Samuelsson, M. Lindh, and E. Larsson. The landscape of viral expression and host gene fusion and adaptation in human cancer. *Nature Communications.*, 2013 Oct 1;4:2513. doi: 10.1038/ncomms3513, 2013.

[3] K. Man, M. Miasari, W. Shi, A. Xin, D. C. Henstridge, S. Preston, M. Pellegrini, G. T. Belz, G. K. Smyth, M. A. Febbraio, S. L. Nutt, and A. Kallies. The transcription factor IRF4 is essential for TCR affinity-mediated metabolic programming and clonal expansion of T cells. *Nature Immunology*, 2013 Sep 22. doi: 10.1038/ni.2710, 2013.

[4] L. Spangenberg, P. Shigunov, A. P. Abud, A. R. Cofr, M. A. Stimamiglio, C. Kuligovski, J. Zych, A. V. Schittini, A. D. Costa, C. K. Rebelatto, P. R. Brofman, S. Goldenberg, A. Correa, H. Naya, and B. Dallagiovanna. Polysome profiling shows extensive posttranscriptional regulation during human adipocyte stem cell differentiation into adipocytes. *Stem Cell Research*, 11:902–12, 2013.

[5] J. Z. Tang, C. L. Carmichael, W. Shi, D. Metcalf, A. P. Ng, C. D. Hyland, N. A. Jenkins, N. G. Copeland, V. M. Howell, Z. J. Zhao, G. K. Smyth, B. T. Kile, and W. S. Alexander. Transposon mutagenesis reveals cooperation of ETS family transcription factors with signaling pathways in erythro-megakaryocytic leukemia. *Proc Natl Acad Sci U S A*, 110:6091–6, 2013.

[6] B. Pal, T. Bouras, W Shi, F. Vaillant, J. M. Sheridan, N. Fu, K. Breslin, K. Jiang, M. E. Ritchie, M. Young, G. J. Lindeman, G. K. Smyth, and J. E. Visvader. Global changes in the mammary epigenome are induced by hormonal cues and coordinated by Ezh2. *Cell Reports*, 3:411–26, 2013.

[7] Y. Liao, G. K. Smyth, and W. Shi. featureCounts: an efficient general-purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30:923–30, 2014.

[8] SEQC/MAQC-III Consortium. A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium. *Nature Biotechnology*, 32:903–14, 2014.

[9] Y. Liao, G. K. Smyth, and W. Shi. ExactSNP: an efficient and accurate SNP calling algorithm. *In preparation.*