Subread/Rsubread Users Guide

Subread v1.3.5-p4/Rsubread v1.10.3

21 Jun 2013

Wei Shi and Yang Liao

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

Contents

Chapter 1

Introduction

This document describes in details the programs included in the `Subread` software package. The `Subread` package includes a suite of programs that are useful in processing next-generation sequencing data. The programs have the functions of read alignment, junction detection, indel detection, read summarization

`Subread` itself is a superfast, sensitive and accurate read aligner[1], which is designed to align reads generated from the 2nd and 3rd generation sequencers to a reference genome . It supports a variety of sequencing platforms including Illumina GA/HiSeq, ABI SOLiD, Life Science 454, Helicos Heliscope and Ion Torrent. It can align short reads, long reads and reads of variable lengths. `Subread` has been shown to be very accurate in mapping Illumina and 454 data [2, 3]

`Subread` employs a mapping paradigm called "seed-and-vote" [1], which is fundamentally different from the "seed-and-extend" paradigm used by many read aligners. The "seed-and-vote" paradigm extracts a number of subreads (16 mers) from the read and then uses these subreads to vote for the mapping location of the read, rather than performing an computational expensive extension operation to determine the location of the read like "seed-and-extend" does. The power and flexibility of the new mapping paradigm enables `Subread` to achieve a superior efficiency without losing accuracy and sensitivity. This paradigm is especially powerful for the alignment of RNA-seq data because it can automatically determines if the reads should be globally aligned or locally aligned. This paradigm also enables a highly accurate detection of insertions and deletions, via using perfectly matched subreads flanking the indels to call them.

The `subjunc` program included in this package is designed to detect exon-exon junctions and to perform full alignments for RNA-seq reads. It also take advantage of the powerful "seed-and-vote" paradigm to achieve a highly accurate junction detection and read mapping [1]. It returns a BED file which includes the chromosomal locations of exon-exon junctions and the number of reads supporting these junctions. It also outputs a SAM file which records the detailed mapping results for reads.

The `featureCounts` program is designed to assign mapped reads or fragments (for 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[4].

Many of the programs inclued in the `Subread` package have also been implemented in the Bioconductor package `Rsubread`, providing R users easy access to these programs via their fimiliar programming environment. The `Rsubread` package can be downloaded from: `http://bioconductor.org/packages/2.10/bioc/html/Rsubread.html`.

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://nar.oxfordjournals.org/content/early/2013/04/03/nar.gkt214.abstract

If you use featureCounts, please cite:

> Liao Y, Smyth GK and Shi W. featureCounts: an efficient general-purpose read summarization program. arXiv:1305.3347, 2013
> http://arxiv.org/abs/1305.3347

2.2 Download and installation

2.2.1 Subread package

Download the Subread package from http://subread.sourceforge.net and type the following command to uncompress it:

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

Enter the `src` subdirectory under the home directory of this package and then issue the following command to install it on a Linux/unix computer:

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

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

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

A new subdirectory called `bin` will be created under the home directory of the software package, and the executables generated from the compilation will be saved to that subdirectory. 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

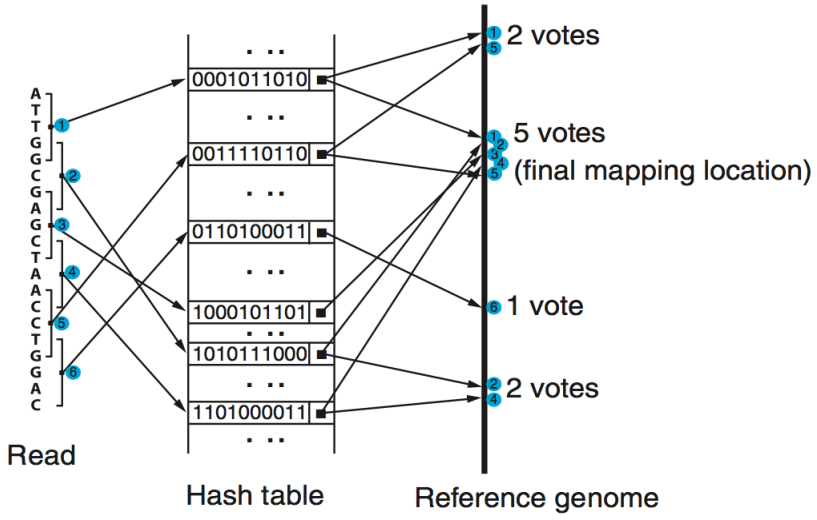
Bioconductor mailing list (`http://bioconductor.org/`) and SeqAnswer forum (`http://www.seqanswers.com`) are the best places to get help and report bugs. Alternatively, you may contact Wei Shi (shi at wehi dot edu dot au) directly.

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.

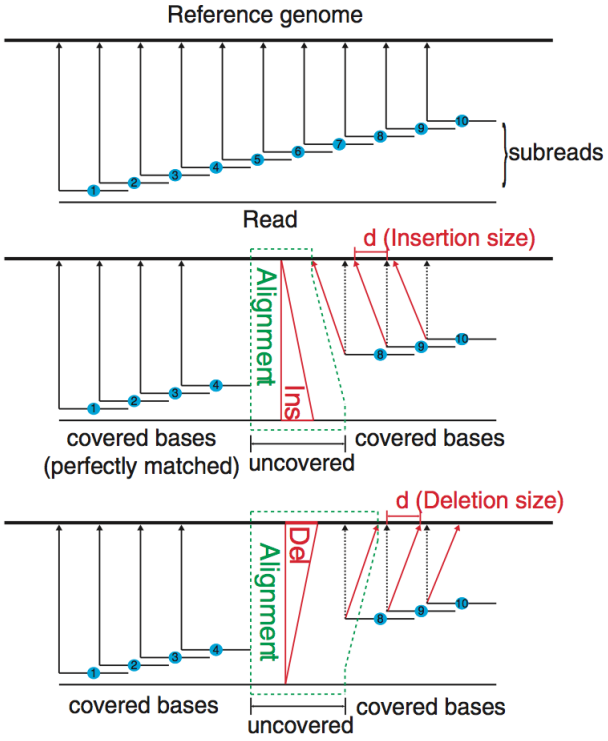


Two aligners have been developed under the seed-and-vote paradigm, including `Subread` and `Subjunc`. `Subread` is a general-purpose read aligner, which can be used to map both genomic DNA-seq and RNA-seq read data. Its running time is determined by the number of *subreads* extracted from each read, 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 Indel detection



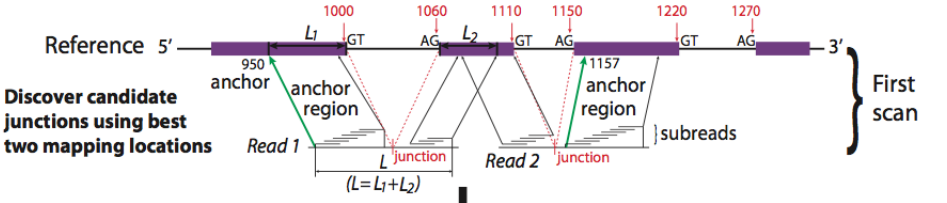
The seed-and-vote paradigm is very powerful in detecting indels (insertions and deletions). The figure below shows how we use the *subreads* to confidently detect indels. When there is an indel existing in a read, mapping locations of subreads extracted after the indel will be shifted to the left (insertion)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

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 and receptor 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.

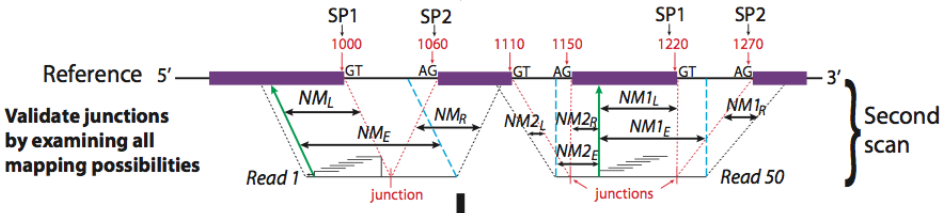


Putative junctions

Chr	Splicing Point 1 (SP1)	Splicing Point 2 (SP2)
1	1000	1060
1	1100	1150
1	1220	1270
•••	•••	•••
10	500	10,000
•••	•••	•••

Anchor information

Reads	Chr	Location in genome	Location in read	Region length
Read 1	1	950	5	46
Read 2	1	1157	44	52
•••	•••	•••	•••	•••
Read 50	1	1168	26	46
•••	•••	•••	•••	•••



Validated junctions

Chr	SP1	SP2	# supporting reads	
•••	•••	•••	•••	
1	1000	1060	20	✓
1	1110	1150	22	✓
1	1220	1270	20	✓
•••	•••	•••	•••	
10	500	10,000	0	X
•••	•••	•••	•••	

Read mapping results

Reads	Chr	Location	CIGAR
Read 1	1	945	55M50N45M
Read 2	1	1073	38M30N62M
•••	•••	•••	•••
Read 50	1	1102	9M40N70M50N5M1D16M
•••	•••	•••	•••

Chapter 4

Mapping reads generated by genomic DNA sequencing technologies

4.1 A quick start for SourceForge Subread

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

Step 1: Building 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: Aligning the reads

Map single-end reads using 5 threads:
`subread-align -T 5 -i my_index -r reads.txt -o subread_results.sam`

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 Bioconductor Rsubread

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",output_file="rsubread.sam",nthreads=5)
```

Detect indels of up to 16bp:

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

Report up to three best mapping locations:

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

Report uniquely mapped reads only:

```
align(index="my_index",readfile1="reads.txt",output_file="rsubread.sam",unique=TRUE)
```

Map paired-end reads:

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

4.3 Index building

The `subread-buildindex` (`buildindex` function in Rsubread) program builds an base-space or color-space index using the reference sequences. The reference sequences should be in FASTA format (the header line for each chromosomal sequence starts with ">").

This program extracts all the 16 mer sequences from the reference genome at a 2bp interval and then uses them to build a hash table. Keys in the hash table are unique 16 mers and values are their chromosomal locations. 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 parathesis in the first column are used by `buildindex`.

Arguments	Description
-o < *basename* > (`basename`)	Specify the base name of the index to be created.
-f < *int* > (`TH_subread`)	Specify the threshold for removing uninformative subreads (highly repetitive 16mers). Subreads will be excluded from the index if they occur more than threshold number of times in the reference genome. Default value is 24.
-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.
-c (`colorspace`)	Build a color-space index.
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.

4.4 Read mapping

The `subread-align` program (`align` in Rsubread) 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. `subread-align` program automatically determines if a read should be globally aligned or locally aligned, making it particularly poweful in mapping RNA-seq reads. Table 2 describes the arguments used by the `subread-align` program (`align` function in Rsubread).

Table 2: arguments used by the `subread-align` program included in the SourceForge Subread package and `align` function in the Rsubread package. Arguments in parathesis in the first column are used by `align` function in Rsubread.

Arguments	Description
-i < *index* > (`index`)	Specify the base name of the index.
-r < *input* > (`readfile1`)	Give the name of an input file(FASTQ/FASTA format). For paired-end read data, this gives the first read file and the other read file should be provided via the -R option.
-o < *output* > (`output_file`)	Give the name of the output file (SAM format).
-n < *int* > (`nsubreads`)	Specify the number of subreads extracted from each read, 10 by default.
-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.
-T < *int* > (`nthreads`)	Specify the number of threads/CPUs used for mapping. 1 by default.
-I < *int* > (`indels`)	Specify the number of INDEL bases allowed in the mapping. 5 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.
-u (`unique`)	Output the uniquely mapped reads only.
-B < *int* > (`nBestLocations`)	Specify the maximal number of equally-best mapping locations allowed to be reported for a read. Its value has to be within the range of 1 to 16. The default value is 1. The number of equally-best locations reported for a read will be less than or equal to the specified value. For example, if a read has two equally-best mapping locations, but the 'B' was set to 5, then only two locations will be reported for this read.
-Q (codetieBreakQS)	Use mapping quality scores to break ties when more than one best mapping locations are found.
-H (`tieBreakHamming`)	Use Hamming distance to break ties when more than one best mapping locations are found.

-J (`markJunctionReads`)	Mark those bases which can not be aligned together with other bases from the same read using the 'S' operation in the CIGAR string (soft-clipping). This option is useful for marking exon-spanning reads and fusion reads. For RNA-seq data, 'Subjunc' program shall then be used to perform full alignments for these reads.
-R < *input* > (`readfile2`)	Provide the name of the second reads file from paired-end data. The program will then be switched to paired-end read mapping mode.
-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.
-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.
-S < $ff : fr : rf$ > (`PE_orientation`)	Specify the orientation of two reads from the same pair, 'fr' by default (first read is located on the forward strand and second read on the reverse strand).
-G < *int* > (`DP_GapOpenPenalty`)	Specify the penalty for opening a gap when applying the Smith-Waterman dynamic programming to detecting indels. -2 by defaut.
-E < *int* > (`DP_GapExtPenalty`)	Specify the penalty for extending the gap when performing the Smith-Waterman dynamic programming. 0 by defaut.
-X < *int* > (`DP_MismatchPenalty`)	Specify the penalty for mismatches when performing the Smith-Waterman dynamic programming. 0 by defaut.
-Y < *int* > (`DP_MatchScore`)	Specify the score for the matched base when performing the Smith-Waterman dynamic programming. 2 by defaut.
-b	Output base-space reads instead of color-space reads in the SAM output. This option is only applicable for color-space read mapping.

4.5 Mapping quality scores

Both Subread and Subjunc aligners output a mapping quality score for each mapped read, defined by

$$\mathrm{MQS} = 100 + \frac{100}{l}\left\{\sum_{i \in b_m}(1-p_i) \; - \sum_{i \in b_{mm}}(1-p_i)\right\}$$

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 available in the FASTQ file (raw read data file). High quality bases have low base-calling p values. Read bases which were found to be insertions are treated as matched bases in the MQS calculation. The MQS is a read-length normalized value, which is in the range of 0 to 200. If a read can be best mapped to more than one location, its MQS will be divided by the number of such locations.

Chapter 5

Mapping reads generated by RNA sequencing technologies

5.1 A quick start for SourceForge Subread

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

For the purpose of differential expression analysis (ie. discovering differentially expressed genes), we recommend you to use the Subread aligner. 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 be 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 6.5), and differential expression analysis can be readily performed using the read summarization results (they include the summarization results from summarizing reads falling within exons as well). Therefore, Subread is sufficient for the read alignments when the purpose of the RNA-seq analysis is to perform a differential expression analysis. Also, Subread could report more mapped reads than Subjunc because, for example, exon-spanning reads may not be accepted by Subjunc if they contain non-canonical splicing signals (donor/receptor sites), but they will be reported by Subread as long as they have a good match with the target region.

For other purposes of the RNA-seq data anlayses such as exon-exon junction detection and genomic mutation detection, in which the reads need to be fully aligned, the Subjunc aligner should be used. Below is an example of using Subjunc to perform global alignments for paired-end RNA-seq reads. Note that there are two files included in the alignment output: one file including the discovered exon-exon junctions and the other including the alignment results for the reads (a SAM file).

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

5.2 A quick start for Bioconductor Rsubread

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 an explanation of which aligner should be used for mapping your RNA-seq data. To use the Subread aligner to map your data, you can use the following R command (mapping a single-end RNA-seq dataset). Options of the `align` function can be found in its help page in the Rsubread package.

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

To use the Subjunc aligner to map your data, you can use the following R commands (mapping a paired-end RNA-seq dataset). Note that you have to run `align` first before you can run `subjunc`. Options of the `subjunc` function can be found in its help page in the Rsubread package.

```
align(index="my_index",readfile1="rnaseq-reads1.txt",readfile2="rnaseq-reads2.txt",
output_file="subread_results.sam")
subjunc(index="my_index",samfile="subread_results.sam",output_file="subjunc_results.sam",
paired_end=TRUE)
```

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 exon-spanning reads and reports the mapping locations that have the largest overlap with such reads. These reads can then be readily assigned to genes for the expression analysis. For gene expression analysis, global alignments for the exon-spanning reads are not required because the local aligments are sufficient to get reads accurately assigned to genes.

However, global alignments are required when the purpose of the RNA-seq data analysis is to discover exon-exon junctions, to detect genomic mutations in the RNA-seq data or to perform allele-specific gene expression analysis. 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 find the reference regions matching the segments within the read, and then using the splicing signals (donor and receptor sites) to precisely determine the mapping locations of the read bases.

This program takes as input either a raw read data file (FASTQ/FASTA format) or a SAM file (e.g. SAM output from Subread aligner). Note that the `subjunc` function included in the Rsubread package only accepts SAM input. The output of Subjunc aligner includes a list of discovered exon-exon junction locations and also the complete alignment results for reads. Table 3 describes the arguments used by the Subjunc program.

Table 3: Arguments used by the `subjunc` program included in the SourceForge Subread package and the `subjunc` function in the Rsubread package. Arguments used by the `subjunc` function in Rsubread are included in parathesis in the first column.

Arguments	Description
-i < *index* > (`index`)	Specify the base name of the index.
-r < *input* >	Give the name of an input file(FASTQ/FASTA format). Both base-space and color-space read data are supported. For paired-end read data, this gives the first read file and the other read file should be provided via the -R option. Note that the `subjunc` function in the Rsubread package does not use this parameter.)
-o < *output* > (`output_file`)	Give the name of the output file.
-n < *int* > (`nsubreads`)	Give the number of subreads extracted from each read. 14 by default.
–singleSAM < *input* > (`samfile`)	Use as input a SAM file that includes mapping results for single-end reads (eg. output of the Subread aligner).
–pairedSAM < *input* > (`samfile`, `paired_end=TRUE`)	Use as input a SAM file that includes mapping results for paired-end reads (eg. output of the Subread aligner). Note that for the `subjunc` function in the Rsubread package, its parameter `paired_end` needs to be set to `TRUE` when paired-end SAM file is provided.
-T < *int* > (`nthreads`)	Specify the number of threads/CPUs used for mapping. 1 by default.
-I < *int* > (`indels`)	Specify the number of INDEL bases allowed in the mapping. 5 by default.
-P < 3 : 6 > (`nsubreads`)	Specify the format of Phred scores used in the input data, '3' for phred+33 and '6' for phred+64. '3' by default. Note that the `subjunc` function in the Rsubread package does not need this parameter because the Phred score offset used in the SAM file is always 33.
-R < *input* >	Provide the name of the second input file in paired-end data. The program will then be switched to paired-end read mapping mode. Note that the `subjunc` function in the Rsubread package does not use this parameter.
-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.

-S $< ff : fr : rf >$ (`PE_orientation`)	Specify the orientation of two reads from the same pair, 'fr' by default (first read is located on forward strand and second read on reverse strand).

Chapter 6

Read summarization

6.1 A quick start for 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.

Summarizing single-end reads using 5 threads:

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

Summarizing BAM format single-end read data:

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

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

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

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

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

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

```
featureCounts -p -B -a annotation.gtf -t exon -g gene_id -i mapping_results_PE.sam
```

```
-o counts.txt
```

Excluding chimeric fragments from the fragment counting:

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

6.2 A quick start for 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)
```

Summarizing single-end reads using 5 threads:

```
featureCounts(files="mapping_results_SE.sam",nthreads=5,annot="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

Summarizing BAM format single-end read data:

```
featureCounts(files="mapping_results_SE.bam",file.type="BAM",annot="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

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

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

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

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE,checkFragLength=TRUE,
minFragLength=50,maxFragLength=600,annot="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

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

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

Excluding chimeric fragments from the fragment counting:

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE,countChimericFragments=FALSE,
annot="annotation.gtf",isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

6.3 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. Care must be taken to ensure that such reads are not over-counted. Here we describe the `featureCounts` program, which is 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 fusions in the reads.
- It takes less than 4 minutes to summarize 20 million pairs of reads to 26k RefSeq genes using one thread, and only uses 40MB of memory (you can run it on a Mac laptop).
- It supports multi-threaded running, making it extremely fast for summarizing large datasets.
- It supports GTF format annotation and SAM/BAM read data.
- It supports strand-specific read summarization.
- It can perform read summarization at both feature level (eg. exons) and meta-feature level (eg. genes).
- 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 discriminates the features, which were overlapped by both ends from the same fragment, from those which were overlapped by only one end so as to get more fragments counted.
- It allows users to specify whether chimeric fragments should be counted.

6.4 Annotation format

A popular annotation format is the GTF annotation format (`http://genome.ucsc.edu/FAQ/FAQformat.html#format4`). `featureCounts` supports the use of this annotation format.

In addition to the GTF format, the `featureCounts` program also supports a simplified annotation format (SAF) which includes the following five columns (tab-delimited):

```
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 -
...
```

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

6.5 featureCounts

featureCounts is a general-purpose read summarization function, which assigns to the genomic features (or meta-features) the mapped reads that were generated from genomic DNA and RNA sequencing.

This function takes as input a set of files containing read mapping results output from a read aligner, and then assigns the mapped reads to genomic features. The acceptable formats of the input files are SAM and BAM.

It can perform read summarization at the feature level or at the meta-feature level (when `-m` is specified). Every entry in the annotation file is a feature. Features are grouped into meta-features by the `gene_id` attribute in a GTF format annotation file, or by the first column of the SAF format annotation file.

A read is said to overlap with a feature when the read and the feature share at least one base. The option `allowMultiOverlap` specifies whether a read is allowed to be assigned to multiple features (counted multiple times) if it overlaps more than 1 feature.

A read is said to overlap with a meta-feature if it overlaps with at least one of the features included in this meta-feature. When summarizing reads at the meta-feature level, a read will only be counted once if it overlaps with a meta-feature even it overlaps with two or more features belonging to that meta-feature. This is useful for summarizing RNA-seq reads for the purpose of expression analysis, because reads should not be counted more than once if they overlap more than exon belonging to the same gene. For the meta-feature level summarization, the option `allowMultiOverlap` controls if a read should be assigned to multiple overlapping meta-features, rather than features.

Table 4 describes the parameters used by the `featureCounts` program.

Table 4: arguments used by the `featureCounts` program included in the SourceForge Subread package and the `featureCounts` function in the Rsubread package. Arguments used by the `featureCounts` function in Rsubread are included in parathesis in the first column.

Arguments	Description
-a < *input* > (`annot`)	Give the name of the annotation file. The program assumes that the provided annotation file is in GTF format. Use `-F` option to specify other annotation formats.
-F (`isGTFAnnotationFile`)	Specify the format of the annotation file. Acceptable formats include 'GTF' and 'SAF' (see Section 6.4 for details). 'GTF' by default. The `featureCounts` function in Rsubread uses parameter `isGTFAnnotationFile` to specify whether the provided annotation is in GTF format or not.
-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.
-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.
-i < *input* > (`files`)	Give the name of an input file including the read mapping results. The file should have a SAM or BAM format. `-b` option needs to be specified if the file is in BAM format. Note that `featureCounts` function in Rsubread accepts more than one input file.
-b (`file.type`)	Indicate the input file is in BAM format. The parameter `file.type`, used by the `featureCounts` function in Rsubread, has two possible values including `SAM` and `BAM` (the default value is `SAM`).
-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.
-f (`useMetaFeatures`)	If specified, read summarization will be performed at the feature level. By default (`-f` is not specified), the read summarization is performed at the meta-feature level.

-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 matched feature if `-f` is specified). In this case, reads (or fragments) will be counted more than once if they overlap multiple meta-features (or features).
-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, the strand of the first read is taken as the strand of the whole fragment. The FLAG field in the SAM/BAM file is used to tell if a read is the first read in the fragment.
-M	If specified, multi-mapping reads/fragments will be counted (ie. 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.
-Q < *int* > (`minMQS`)	The minimum mapping quality score a read must have so as to be counted. For paired-end reads, at least one end should satisfy this criteria. 0 by default.
-T < *int* > (`nthreads`)	Number of the threads. 1 by default.
-R	Output the read summarization results for each read. Note that this option is not supported by `featureCounts` function in Rsubread.
-p (`isPairedEnd`)	If specified, fragments (or templates) will be counted instead of reads. This option is only applicable for paired-end reads. The two reads from the same fragment must be adjacent to each other in the provided SAM/BAM file.
-P (`checkFragLength`)	If specified, the fragment length will be checked when assigning fragments to meta-features or features. This option should be used together with `-p` (or `isPairedEnd` in Rsubread `featureCounts`). The fragment length thresholds should be specified using `-d` and `-D` options.
-d < *int* > (`minFragLength`)	Minimum fragment/template length, 50 by default.
-D < *int* > (`maxFragLength`)	Maximum fragment/template length, 600 by default.
-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`).

Chapter 7

Case studies

7.1 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. A_1 and A_2 are both Universal Human Reference RNA (UHRR) samples but they underwent separate sample preparation. B_1 and B_2 are both Human Brain Reference RNA (HBRR) samples and they also underwent separate sample preparation. Note that these libraries only included reads originating from human chromosome 1 (according to Subread aligner). These read data were generated by the SEQC Consortium. We have put into a tar ball these read data and the reference sequence data of chromosome 1 from human genome build GRCh37/hg19, and it can be downloaded from
`http://bioinf.wehi.edu.au/RNAseqCaseStudy/data.tar.gz` (283MB).

After downloading the dataset, uncompress it and save them to your current working directory. Launch R and load Rsubread and limma libraries using the following commands.

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

If these libraries have not been installed in your R, check Bioconductor website for instructions on the package installation. It will be best to use the latest version of R. This case study was tested on R version 3.0.0. Note that Rsubread can only be installed on Linux/Unix and Mac OS X computers.

Index building. Build an index for human chromosome 1. This will take ∼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 will take ∼4 minutes. SAM files which include the mapping results will be generated in your current working directory.

```
for(i in c("A_1","A_2","B_1","B_2"))
  align(index="chr1",readfile1=paste(i,"txt",sep="."),output_file=paste(i,"sam",sep="."),
  unique=TRUE,indels=5)
```

Read summarization. Summarize mapped reads to RefSeq genes. This will take less than half a minute. Note that the featureCounts function has built-in annotation for Refseq genes. featureCounts returns an R 'List' object which can be directly fed into limma for normalization and differential expresssion analysis. This object includes raw read count for each gene in each library and also annotation information.

```
counts <- featureCounts(files=c("A_1.sam","A_2.sam","B_1.sam","B_2.sam"),genome="hg")
```

Filtering. Calculate RPKM (reads per kilobases of exon per million reads mapped) values for genes and use these values to filter out those genes which failed to achieve a 0.5 RPKM in at least two libraries.

```
counts_rpkm <- apply(counts$counts,2,function(x) x*(1000/counts$annotation$Length)*(1e6/sum(x)))
isexpr <- rowSums(counts_rpkm >= 0.5) >= 2
x <- counts$counts[isexpr,]
```

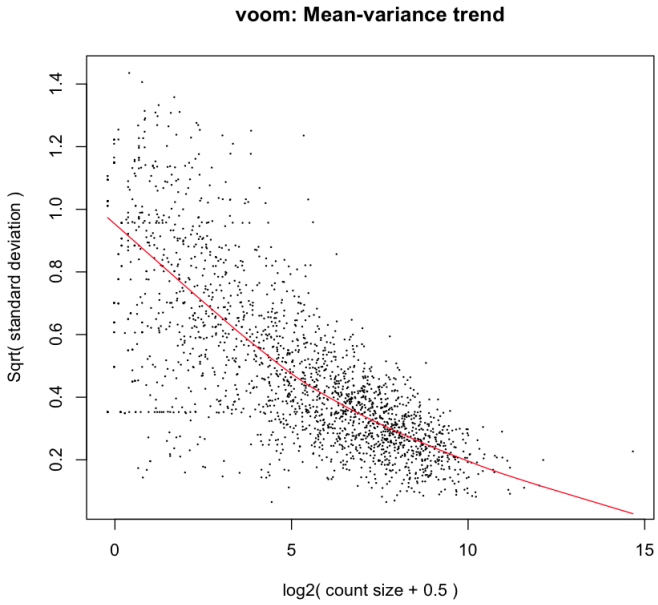
Design matrix. The following analyses are very similar to the analyses performed for microarray expression data. Firstly, we create a design matrix:

```
celltype <- factor(c("A","A","B","B"))
design <- model.matrix(~0+celltype)
colnames(design) <- levels(celltype)
```

Normalization. Then we perform voom normalization:

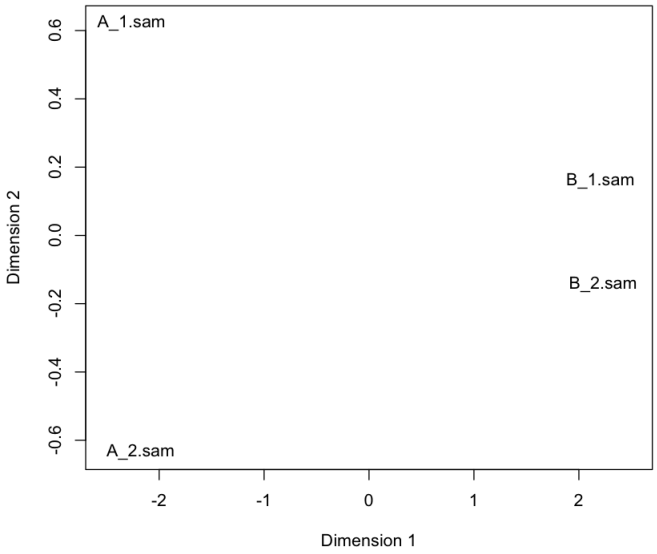
```
y <- voom(x,design,plot=TRUE)
```

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



Sample clustering. The following multi-dimensional scaling 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 the eBayes moderated t statistic. Here we compare A vs B. 560 and 994 genes were found down- and up-regulated in sample A compared to sample B, respectively.

```
fit <- lmFit(y,design)
contr <- makeContrasts(AvsB=A-B,levels=design)
```

```
fit.contr <- eBayes(contrasts.fit(fit,contr),trend=TRUE)
dt <- decideTests(fit.contr)
summary(dt)
   AvsB
-1  560
0   497
1   994
```

List top 10 differentially expressed genes:

```
options(digits=3)
topTable(fit.contr)
            ID logFC AveExpr     t  P.Value adj.P.Val    B
1905 100131754 -1.62    16.0 -77.3 2.02e-17  4.14e-14 28.9
200       2752 -2.38    12.9 -59.5 4.45e-16  4.56e-13 27.5
319       4904  3.00    11.5  54.3 1.31e-15  8.41e-13 26.4
144       2023  2.72    13.4  53.3 1.64e-15  8.41e-13 25.9
415       6135  2.24    12.1  51.6 2.40e-15  9.64e-13 25.8
418       6202  2.40    12.1  50.9 2.82e-15  9.64e-13 25.6
770      22883 -2.24    12.5 -50.0 3.51e-15  1.03e-12 25.4
792      23154 -3.73    11.4 -48.7 4.84e-15  1.24e-12 24.9
563       8682 -2.58    11.7 -41.1 3.53e-14  6.95e-12 23.1
414       6125  2.01    11.8  40.9 3.73e-14  6.95e-12 23.0
```

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 Res*, 41:e108, 2013.

[2] 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.

[3] 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 Rep*, 3:411–26, 2013.

[4] Y. Liao, G. K. Smyth, and W. Shi. featureCounts: an efficient general-purpose read summarization program. *arXiv:1305.3347*, 2013.