prebs User Guide

Karolis Uziela, Antti Honkela

October 13, 2014

1 Abstract

The *prebs* package aims at making RNA-sequencing (RNA-seq) data more comparable to microarray data. The comparability is achieved by summarizing sequencing-based expressions of probe regions using a modified version of RMA algorithm (Irizarry et al., 2003). The pipeline takes mapped reads in BAM format as an input and produces either gene expressions or original microarray probe set expressions as an output. A more detailed algorithm description can be found in (Uziela and Honkela, 2013).

2 Installation

prebs can be installed from the the bioconductor using `biocLite` function. This ensures that all of the package dependencies are met.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("prebs")
```

prebsdata package that is needed to run the examples in this vignette is also available from the bioconductor.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("prebsdata")
```

3 Examples

Here we will cover a few simple examples of running *prebs* in two modes: Custom CDF and manufacturer's CDF. The major difference between these two modes is that Custom CDF gives expression values for genes while manufacturer's CDF gives the expression values for the probe sets.

3.1 Loading package and data

To load the package start R and run

```
> library(prebs)
```

The data for our examples is contained in *prebsdata* package. The data package contains two sample BAM files, 3 Custom CDF probe sequence mapping files and 3 manufacturer's CDF probe sequence mapping files. We will use only 2 Custom CDF and 1 manufacturer's CDF probe sequence mapping file in our examples.

The full paths to data files in the *prebsdata* package can be retrieved using `system.file` function.

```
> bam_file1 <- system.file(file.path("sample_bam_files", "input1.bam"),
+                           package="prebsdata")
> bam_file2 <- system.file(file.path("sample_bam_files", "input2.bam"),
+                           package="prebsdata")
> bam_files <- c(bam_file1, bam_file2)
> custom_cdf_mapping1 <- system.file(file.path("custom-cdf",
+     "HGU133Plus2_Hs_ENSG_mapping.txt"), package="prebsdata")
> custom_cdf_mapping2 <- system.file(file.path("custom-cdf",
+     "HGU133A2_Hs_ENSG_mapping.txt"), package="prebsdata")
> manufacturer_cdf_mapping <- system.file(file.path("manufacturer-cdf",
+     "HGU133Plus2_mapping.txt"), package="prebsdata")
```

3.2 Running `calc_prebs` using Custom CDF

The *prebs* package contains only one public function—`calc_prebs`. The most basic usage of `calc_prebs` is running it in Custom CDF mode without parallelization. The default output format of `calc_prebs` is ExpressionSet object defined in *affy* package. The expression values can be accessed using `exprs` function from *affy* package.

```
> prebs_values <- calc_prebs(bam_files, custom_cdf_mapping1)
```

```
Normalizing
Calculating Expression
```

```
> head(exprs(prebs_values))
```

```
                 input1.bam  input2.bam
ENSG00000000003    4.480645    4.035863
ENSG00000000005  -13.313919  -13.313919
ENSG00000000419    4.224720    4.224720
ENSG00000000457    2.611877    4.196825
ENSG00000000460    2.111514    2.300766
ENSG00000000938    2.325804    5.073020
```

Above we can see the expressions of the first few genes with Ensembl gene identifiers. In this example, the expression level of at least one of the genes is negligible (the expression values are in $\log_2$ scale). In fact, most of the other genes that are not shown here also have a negligible expression level, because we designed our sample BAM files so that they contain only mapped reads from the region of the first few genes. Of course, for a real world analysis mapped reads from all of the genes are needed. However, real world BAM files take a lot of disk space, so it was not possible to include them in the sample data set.

Since in this case we did not provide explicit CDF package name, the name was inferred from the probe sequence mapping filename ("`custom-cdf/HGU133Plus2_Hs_ENSG_mapping.txt`" -> *hgu133plus2hsensgcdf*). Both probe sequence mapping file and custom CDF package can be downloaded from Custom CDF website:
`http://brainarray.mbni.med.umich.edu/brainarray/Database/CustomCDF/genomic_curated_CDF.asp`

In particular, this example uses Ensembl custom CDF package for and `HGU133Plus2` platform (version 16.0.0) that can be dowloaded here: `http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/16.0.0/ensg.download/hgu133plus2hsensgcdf_16.0.0.tar.gz`

And the corresponding description archive containing probe sequence mapping file can be downloaded here:
`http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/16.0.0/ensg.download/HGU133Plus2_Hs_ENSG_16.0.0.zip`

3.3 Setting `calc_prebs` output format to a data frame

By default `calc_prebs` outputs an ExpressionSet object with PREBS values. If you prefer to have a data frame as an output, you can set `output_eset` option to FALSE.

```
> prebs_values <- calc_prebs(bam_files, custom_cdf_mapping1, output_eset=FALSE)

Normalizing
Calculating Expression

> head(prebs_values)

  input1.bam input2.bam              ID
1   4.480645   4.035863 ENSG00000000003
2 -13.313919 -13.313919 ENSG00000000005
3   4.224720   4.224720 ENSG00000000419
4   2.611877   4.196825 ENSG00000000457
5   2.111514   2.300766 ENSG00000000460
6   2.325804   5.073020 ENSG00000000938
```

3.4 Running `calc_prebs` with parallelization

Now let's run the same task with a simple parallelization. The results will be identical to the ones above.

```
> library("parallel")
> N_CORES = 2
> CLUSTER <- makeCluster(N_CORES)
> prebs_values <- calc_prebs(bam_files, custom_cdf_mapping1, cluster=CLUSTER)
> stopCluster(CLUSTER)
```

3.5 Running `calc_prebs` for another microarray platform

If we want to run `calc_prebs` with a different microarray platform, we just have to provide another probe sequence mapping file.

```
> prebs_values <- calc_prebs(bam_files, custom_cdf_mapping2)
```

The corresponding Custom CDF package *hgu133a2hsensgcdf* has to be downloaded and installed prior to running this command. It can be found here: `http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/16.0.0/ensg.download/hgu133a2hsensgcdf_16.0.0.tar.gz`

3.6 Running `calc_prebs` using manufacturer's CDF

Running `calc_prebs` with manufacturer's CDF is not so much different either. All we have to do is to provide a suitably formatted probe sequence mapping file.

```
> prebs_values <- calc_prebs(bam_files, manufacturer_cdf_mapping)
```

```
[1] "Finished: /home/biocbuild/bbs-3.0-bioc/R/library/prebsdata/sample_bam_files/input1.ba
[1] "Finished: /home/biocbuild/bbs-3.0-bioc/R/library/prebsdata/sample_bam_files/input2.ba
Normalizing
Calculating Expression
```

```
> head(exprs(prebs_values))
```

```
       input1.bam input2.bam
1007_s -13.307292 -13.307292
1053     3.692921   3.324441
117     -5.147181  -6.009734
121    -13.307292 -13.307292
1255_g -13.307292 -13.307292
1294     2.969080   1.969098
```

As mentioned before, manufacturer's CDF mode gives probe set expressions as an output. In the above example, you can see the the expression values for the first few probe sets of our example data set.

One problem with running `calc_prebs` using manufacturer's CDF is that Affymetrix does not provide probe sequence mappings for most of the microarray platforms.

Therefore, probe sequence mapping files have to be created manually, as it will be discussed in Section 4.

As in Custom CDF case, the CDF package name is inferred from probe sequence mapping file ("`custom-cdf/HGU133Plus2_mapping.txt`" -> *hgu133plus2cdf*). If we are not sure if the mapping file is named correctly, it is better to provide CDF package filename explicitly.

```
> prebs_values <- calc_prebs(bam_files, manufacturer_cdf_mapping,
+                            cdf_name="hgu133plus2cdf")
```

Now we have presented pretty much all important ways of running `calc_prebs` function. From this point, you can proceed with downstream analysis of `calc_prebs` results. However, so far we have left out some important details about input requirements of `calc_prebs` function that will be discussed in the next section.

4 Detailed input specification

The main function of the package `calc_prebs` has the following input arguments:

bam_files	Mapped reads in BAM format
probe_mapping_file, **cdf_name**	Probe sequence mappings in a genome ("`*cdfname*_mapping.txt`" file) and the name of CDF package
cluster	Cluster object for parallelization
output_eset	Option that controls output format (ExpressionSet vs data frame)
paired_ended_reads, **ignore_strand**	Options that control the process of counting reads

In this section we will discuss all the input requirements in more detail. Note that only two input arguments are mandatory: `bam_files` and `probe_mapping_file`. The rest of the arguments are optional and have their default values.

4.1 BAM files

For using `calc_prebs` function you will need to have mapped reads in BAM format. For read mapping we recommend using TopHat software (Trapnell et al., 2009). We suggest to align the reads only to the known transcriptome. You can do this by using `--transcriptome-only` option and supplying your own transcriptome annotation file via `--GTF` option. Transcriptome annotation files can be downloaded from Ensembl FTP server. Finally, we require that reads are mapped to no more than 1 location in the genome. This can be achieved by using option `--max-multihits 1`. So for human genome, sample TopHat run could look like this:

```
tophat --transcriptome-only --max-multihits 1 \
--GTF ./Human_transcriptome/Homo_sapiens.GRCh37.65.gtf \
--transcriptome-index=./Human_transcriptome/known \
--output-dir ./tophat-out hg19 input1.fastq input2.fastq
```

4.2 Probe sequence mappings and CDF packages

`calc_prebs` function can be used in two modes: Custom CDF (Dai et al., 2005) and manufacturer's CDF. Custom CDF mode produces gene expressions while manufacturer's CDF mode produces original probe set expressions. Now we will discuss the input requirements for the two modes in more detail.

4.2.1 Custom CDF

As we have already mentioned `calc_prebs` function requires a probe sequence mapping file and CDF package name as its arguments. For Custom CDF mode, both the mapping file and the package can be downloaded from the Custom CDF website:
`http://brainarray.mbni.med.umich.edu/brainarray/Database/CustomCDF/genomic_curated_CDF.asp`

The Custom CDF supports many types of gene identifiers, but in our examples we are using Custom CDF files with Ensembl gene identifiers (version 16.0.0). In the Custom CDF download page for each microarray platform you can find both the the Custom CDF package file (denoted by "C") and the Custom CDF description archive (denoted by "O") containing the probe sequence mapping file.

The Custom CDF package can be installed like a regular R package (using R CMD INSTALL command). For example, to install hgu133plus2hsensgcdf in Unix-like systems type R CMD INSTALL hgu133plus2hsensgcdf_16.0.0.tar.gz.

The probe sequence mapping file is named as "`*cdfname*_mapping.txt`". Since CDF package name can be inferred from probe sequence mapping filename, explicitly providing CDF package name to `calc_prebs` function is optional. For example, if you are using "`HGU133Plus2_Hs_ENSG_mapping.txt`" probe sequence mapping file do not provide CDF package name, it is assumed that *hgu133plus2hsensgcdf* package is used.

4.2.2 Manufacturer's CDF

The manufacturer's CDF packages can be downloaded and installed from the bioconductor. For example, to install CDF package for `HGU133Plus2` platform, type:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("hgu133plus2cdf")
```

Unfortunately, probe sequence mapping files are not provided for most of the microarray platforms. For some microarray platoforms, such as `HuEx10stv2`, the probe sequence mappings are available from the Affymetrix website (HuEx-1_0-st-v2 Probe Sequences, tabular format). However, they are mapped to an old version of genome assembly (hg16), so we do not recommend using them.

In our data package *prebsdata*, we provide probe sequence mapping files for three microarray platforms: `HGU133Plus2`, `HGU133A2` and `HGFocus`. We have created these files by mapping probe sequences to human genome using Bowtie software Langmead et al. (2009). If you want to use another microarray platform, you will have to map probe sequences yourself. A detailed procedure of creating probe sequence mapping files using Bowtie is outlined below.

For most of the microarray platforms, the probe sequences can be retrieved from the platform's probe package. The probe package name is the same as CDF package name, except that it ends with "probe" instead of "cdf". For example, to install probe package for "hgu133plus2" platform, type:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("hgu133plus2probe")
```

Once you load the *hgu133plus2probe* package, you can find the information about the probe sequences stored in hgu133plus2probe object which can be converted to a data frame.

```
> library("hgu133plus2probe")
> probes <- as.data.frame(hgu133plus2probe)
> head(probes)
```

```
                   sequence    x   y Probe.Set.Name
1 CACCCAGCTGGTCCTGTGGATGGGA  718 317      1007_s_at
2 GCCCCACTGGACAACACTGATTCCT 1105 483      1007_s_at
3 TGGACCCCACTGGCTGAGAATCTGG  584 901      1007_s_at
4 AAATGTTTCCTTGTGCCTGCTCCTG  192 205      1007_s_at
5 TCCTTGTGCCTGCTCCTGTACTTGT  844 979      1007_s_at
6 TGCCTGCTCCTGTACTTGTCCTCAG  537 971      1007_s_at
  Probe.Interrogation.Position Target.Strandedness
1                         3330           Antisense
2                         3443           Antisense
3                         3512           Antisense
4                         3563           Antisense
5                         3570           Antisense
6                         3576           Antisense
```

Next, we should remove rows that have probe set identifiers that start if "AFFX", because these do not target genes and are not relevant to us. Also, we use `xy2indices` function from affy package to convert probe X and Y coordinates to probe IDs and add a new column to the data frame. We will save the resulting data frame to a file "`probes.txt`".

```
> library("affy")
> probes <- probes[substr(probes$Probe.Set.Name,1,4) != "AFFX",]
> probes$Probe.ID <- xy2indices(probes$x, probes$y, cdf="hgu133plus2cdf")
> write.table(probes, file="probes.txt", quote=FALSE, row.names=FALSE, col.names=TRUE)
```

The first column in a file "`probes.txt`" contains probe sequence and the seventh column contains probe ID. To format an input for Bowtie, we need to extract these two columns and format a fasta file:

```
tail -n +2 "probes.txt" | awk '{print ">" $7 "\n" $1 }' > probe_sequences.fa
```

Now we are ready to map the probe sequences to the genome. We suggest using Bowtie options `-a -v 0` to report all perfect match hits. A sample Bowtie run could look like this:

```
bowtie -a -v 0 hg19 -f probe_sequences.fa output_probe_mappings.map
```

After we map probe sequences to the genome, we must convert Bowtie output to the format identical to Custom CDF probe sequence mapping files. The default format of Bowtie output is documented in Bowtie homepage. The first column contains "Read ID" which in our case is "Probe.ID". We have to read Bowtie output file "`output_probe_mappings.map`", and probe sequence information file "`probes.txt`" and merge the two data frames based on "Probe.ID" column. Then, we have to extract the necessary information from the resulting merged table and save it into "`_mapping.txt`" file. Note that we also have to shift Bowtie mapping positions by 1, because it uses a different offset than "`_mapping.txt`" files.

Briefly, here are the commands we have to run:

```
> probe_mappings <- read.table("output_probe_mappings.map")
> colnames(probe_mappings) <- c("Probe.ID", "strand",
+                               "chr", "start", "seq", "match", "multiple")
> # bowtie reports 0-offset, but _mapping.txt files are 1-offset
> probe_mappings$start <- probe_mappings$start + 1
> probes <- read.table("probes.txt", head=TRUE)
> probes <- merge(probes, probe_mappings)
> output_table <- data.frame(Probe.Set.Name=probes$Probe.Set.Name,
+     Chr=probes$chr, Chr.Strand=probes$strand, Chr.From=probes$start,
+     Probe.X=probes$x, Probe.Y=probes$y, Affy.Probe.Set.Name=probes$Probe.Set.Name)
> write.table(output_table, file="HGU133Plus2_mapping.txt",
+             quote=FALSE, sep="\t", row.names=FALSE)
```

The resulting "`_mapping.txt`" file can be used as an input for `calc_prebs`. If some of the probe sequences were mapped to multiple locations, `calc_prebs` function will handle them by summing up the read overlaps from all of these locations. If some probe sequences could not be mapped, `calc_prebs` will assign minimal expression values to these probes. If you are using a manually created "`_mapping.txt`" file, `calc_prebs` will show notifications about the missing probe sequences (that were not mapped) and probe sequences that have duplicates (that were mapped to multiple locations).

4.3 Cluster object for parallel computation

If you have many input BAM files, processing them can be a computationally expensive task. Therefore, *prebs* provides a possibility to parallelize BAM file processing using *parallel* package. In order to parallelize the work, you must use `makeCluster` function to create a cluster object and pass it to `calc_prebs` function. The function `makeCluster` has several parameters that support different types of clusters. For a detailed explanation of `makeCluster`, please, refer to *parallel* package manual. One simple example of using `makeCluster` was already covered in Section 3.

4.4 Output format

`calc_prebs` provides two arguments for output format: ExpressionSet or data.frame. ExpressionSet is a container for high-throughput assays and experimental metadata from Biobase package, whereas data frame is just a standard R data structure.

4.5 Read counting options

`calc_prebs` has a couple of arguments that control the process of the read counting. `paired_ended_reads` argument ensures the correct treatment of paired-ended reads. If your data contains paired-ended reads, you should set this option to `TRUE`, otherwise the two mate reads will be treated as independent units. Another argument, `ignore_strand` controls whether the strand from which the reads comes should be considered during read-counting. If your data comes from strand-specific RNA-seq protocol, set this option to FALSE, otherwise, leave it at its default value (TRUE).

5 Session Info

```
> sessionInfo()

R version 3.1.1 Patched (2014-09-25 r66681)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base
```

```
other attached packages:
 [1] hgu133plus2probe_2.15.0 AnnotationDbi_1.28.0    hgu133plus2cdf_2.15.0
 [4] prebs_1.6.0             affy_1.44.0             Biobase_2.26.0
 [7] GenomicAlignments_1.2.0 Rsamtools_1.18.0        Biostrings_2.34.0
[10] XVector_0.6.0           GenomicRanges_1.18.0    GenomeInfoDb_1.2.0
[13] IRanges_2.0.0           S4Vectors_0.4.0         BiocGenerics_0.12.0

loaded via a namespace (and not attached):
 [1] BBmisc_1.7            BatchJobs_1.4         BiocInstaller_1.16.0
 [4] BiocParallel_1.0.0    DBI_0.3.1             RSQLite_0.11.4
 [7] affyio_1.34.0         base64enc_0.1-2       bitops_1.0-6
[10] brew_1.0-6            checkmate_1.4         codetools_0.2-9
[13] digest_0.6.4          fail_1.2              foreach_1.4.2
[16] iterators_1.0.7       preprocessCore_1.28.0 sendmailR_1.2-1
[19] stringr_0.6.2         tools_3.1.1           zlibbioc_1.12.0
```

References

Manhong Dai, Pinglang Wang, Andrew D Boyd, Georgi Kostov, Brian Athey, Edward G Jones, William E Bunney, Richard M Myers, Terry P Speed, Huda Akil, Stanley J Watson, and Fan Meng. Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data. *Nucleic Acids Res*, 33 (20):e175, 2005. doi: 10.1093/nar/gni179.

Rafael A Irizarry, Bridget Hobbs, Francois Collin, Yasmin D Beazer-Barclay, Kristen J Antonellis, Uwe Scherf, and Terence P Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, Apr 2003. doi: 10.1093/biostatistics/4.2.249.

Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009. doi: 10.1186/gb-2009-10-3-r25.

Cole Trapnell, Lior Pachter, and Steven L Salzberg. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, May 2009. doi: 10.1093/bioinformatics/btp120.

Karolis Uziela and Antti Honkela. Probe region expression estimation for rna-seq data for improved microarray comparability. April 2013. arXiv:1304.1698 [q-bio.GN].