BUSseq: Batch Effects Correction with Unknown Subtypes for scRNA-seq data User's Guide

Fangda Song, Ga Ming Chan and Yingying Wei*

The Chinese University of Hong Kong

*sfd1994895@gmail.com

March 22, 2020

Contents

BUSseq: Batch Effects Correction with Unknown Subtypes for scRNA-seq data User's Guide

1 Introduction

Single-cell RNA-sequencing (scRNA-seq) technologies enable the measurement of the transcriptome of individual cells, which provides unprecedented opportunities to discover cell types and understand cellular heterogeneity [1]. Despite their widespread applications, single-cell RNA-sequencing (scRNA-seq) experiments are still plagued by batch effects and dropout events.

One of the major tasks of scRNA-seq experiments is to identify cell types for a population of cells [1]. Therefore, the cell type of each individual cell is always unknown and is the target of inference. However, most existing methods for batch effects correction, such as Combat space [2] and the surrogate variable analysis (SVA)([3], [4]), are designed for bulk experiments and require knowledge of the subtype information, which corresponds to cell type information for scRNA-seq data, of each sample a priori.

Here, the R package *BUSseq* fits an interpretable Bayesian hierarchical model—the Batch Effects Correction with Unknown Subtypes for scRNA seq Data(BUSseq)—to correct batch effects in the presence of unknown cell types [5]. BUSseq is able to simultaneously correct batch effects, clusters cell types, and takes care of the count data nature, the overdispersion, the dropout events, and the cell-specific sequencing depth of scRNA-seq data. After correcting the batch effects with BUSseq, the corrected value can be used for downstream analysis as if all cells were sequenced in a single batch. BUSseq can integrate the read count matrices measured from different platforms and allow cell types to be measured in some but not all of the batches as long as the experimental design fulfills the conditions listed in [5].

This guide provides step-by-step instructions for applying the BUSseq model to correct batch effects and identify the unknown cell type indicators for each cell for scRNA-seq data.

2 Data Preparation

The input data should be an R list with the length equal to the number of batches. Each element of the list is a read count matrix, where each row represents a gene and each column corresponds to a cell. Specifically, assuming there are three batches, the R list consists of three read count matrices with genes in rows and cells in columns. In the following, we provide

examples to illustrate the input formats, where `BUSseqfits_example` is a sample `BUSseqfits` object. We use the data list stored in this object as an example.

```
library(BUSseq)

#Input data is a list
CountData <- BUSseqfits_example$CountData_raw
class(CountData)

## [1] "list"

#The length of the list is three, so we have three batches
length(CountData)

## [1] 3

#Each element of the list is a matrix
class(CountData[[1]])

## [1] "matrix"

#In the matrix, each row is a gene, and each column corresponds to a cell
dim(CountData[[1]])

## [1] 1000  150

dim(CountData[[2]])

## [1] 1000  150

dim(CountData[[3]])

## [1] 1000  150

#Peek at the read counts
head(CountData[[1]][,1:4])

##      [,1] [,2] [,3] [,4]
## [1,]   15   31   27    0
## [2,]   16   13   20   30
## [3,]   38   22   14   14
## [4,]   19    0   16   25
## [5,]    0    9   29   14
## [6,]   30   14   28    0
```

The example data `CountData` consist of three batches. In total, 1,000 genes are measured. The number of cells in each batch is 150, respectively. Because it is a simulated dataset, we actually know that all of the cells come from 4 cell types.

In a nutshell, the user can use the R list with length equal to the batch number as input. Note that the gene numbers of all batches need to be the same.

3 Model Fitting

Once we have prepared the input data and specified the cell type number, we are able to fit the BUSseq model, which requires the function `BUSseq_MCMC`.

The first argument, `ObservedData`, of `BUSseq_MCMC` should be an R list where each element is a data matrix for a specific batch. In the matrix, each row corresponds to a gene or a genomic feature and each column corresponds to a cell.

The second argument, `seed`, lets the user obtain reproducible results.

The third argument, `n.celltypes`, is the number of cell types among all cells, which needs to be specified by the user in advance. As discussed later, if `n.celltypes` is unknown, we can vary the cell type number and use the Bayesian Information Criterion (BIC) to select the optimal number.

The forth argument, `n.iterations`, is the total number of iterations of the MCMC algorithm for the posterior inference of the BUSseq model. The user can also set the number of burnin iterations by the argument, `n.burnin`. Given `n.iterations`, the default number of burnins is `n.iterations`/2 iterations. The parameters are inferred by samples after the burnin iterations.

```
  #Conduct MCMC sampling and posterior inference for BUSseq model
  BUSseqfits_res <- BUSseq_MCMC(ObservedData = CountData, seed = 1234, n.celltypes = 4,
                    n.iterations = 500)
##    conducting the posterior sampling...
```

```
##    The MCMC sampling takes:  1.046 mins
##    conducting the posterior inferences...
##    calculating posterior means and posterior takes:  0.04 mins
  class(BUSseqfits_res)
## [1] "BUSseqfits"
```

The `summary` command provides an overview of the output object `BUSseqfits_res` from `BUSseq_MCMC`. `BUSseqfits_res` collects two lists of the read count matrices and all posterior estimates of parameters as well as the BIC value. These two lists consist of the raw observed read counts, the inferred underlying true read counts after imputing the dropout events. The posterior estimates contain the cell-type indicators for each cell, the cell-type proportions for each batch, the cell-type-specific mean expression levels, the location batch effects, the overdispersion parameters and the odds ratios of the logistic regression for dropout events.

```
  summary(BUSseqfits_res)
## B=3 batches
## G=1000 genes
## K=4 cell types
## N=450 cells in total
## Run 500 iterations with the first 250 iterations as burnin in the MCMC algorithm.
## BUSseqfits is an R list that contains the following main elements:
##    BUSseqfits$w.est :  the estimated cell type indicators, a list with length
equal to B.
##    BUSseqfits$pi.est :  the estimated cell type proportions across batches,
a K by B matrix.
##    BUSseqfits$gamma.est :  the estimated the coefficients of the logistic regression
for the dropout events, a B by 2 matrix
##    BUSseqfits$alpha.est :  the estimated log-scale baseline expression levels,
a vector with length G.
```

```
##    BUSseqfits$beta.est :  the estimated cell type effects, a G by K matrix.
##    BUSseqfits$delta.est :  the estimated cell-specific effects, a list with
length equal to B.
##    BUSseqfits$nu.est :  the estimated location batch effects, a G by B matrix.
##    BUSseqfits$phi.est :  the estimated overdispersion parameters, a G by B matrix.
##    BUSseqfits$BIC : the BIC, a scalar.
##    BUSseqfits$D.est :  the intrinsic gene indicators, a vector with length N.
##    For more output values, please use "?BUSseq_MCMC"
##
```

4 Estimated Cell Types, Batch and Cell-Specific Effects

Our main interests are the estimation of the cell type for each cell and the estimation of the batch effects. We can call the `celltypes` function to extract the cell type information from `BUSseqfits_res`.

```
  celltyes_est <- celltypes(BUSseqfits_res)
## Batch 1 cells' cell type indicators:  1,1,1...  ...
## Batch 2 cells' cell type indicators:  1,1,1...  ...
## Batch 3 cells' cell type indicators:  1,1,1...  ...
## The output format is a list with length equal to the batch number.
## Each element of the list is a cell type indicator vector in that batch.
```

There is a message from the function `celltypes` to remind the user of the format of `celltyes_est`. In this example, `celltypes_est` is a list of length three, corresponding to the three batches in the study. `celltypes_est[[1]]` shows the cell type for each of the 150 cells in batch one.

Similarly, you can call `location_batch_effects` and `overdispersions` functions to get the estimated location batch effects and batch-specific gene-specific overdispersion parameters. Note that the first batch is taken as the reference batch, so its location batch effects are zeros for all genes.

```
  location_batch_effects_est <- location_batch_effects(
                                                BUSseqfits_res)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a batch.

  head(location_batch_effects_est)

##      [,1]     [,2]     [,3]
## [1,]    0 1.916466 3.141426
## [2,]    0 1.921966 3.161098
## [3,]    0 1.888274 3.097430
## [4,]    0 1.969050 3.122910
## [5,]    0 1.865936 3.084606
## [6,]    0 1.975787 3.130177

  overdispersion_est <- overdispersions(BUSseqfits_res)

## The output format is a matrix.
##
## Each row represents a gene, and each column corresponds to a batch.

  head(overdispersion_est)

##          [,1]      [,2]      [,3]
## [1,] 9.951195 11.033341 10.553404
## [2,] 8.842701 10.797923 13.380978
## [3,] 9.456289  9.452470 10.316428
## [4,] 7.938066  8.296613 11.260732
## [5,] 8.338051 12.153549  9.002212
## [6,] 7.255012  9.808790  7.730292
```

The cell-specific size effects is available using the `cell_effect_values`. Note that the first element of each batch is 0 as the first cell in each batch is taken as the reference one.

```
  cell_effects_est <- cell_effect_values(BUSseqfits_res)

## The output format is a list with length equal to the batch number.

## Each element of the list is a cell-specific size factor vector of that batch.

  head(cell_effects_est[[1]])

## [1]  0.00000000 -0.02263242  0.01530716  0.01355882  0.02199769 -0.03120789
```

`celltype_mean_expression` function provides the estimated cell-type-specific mean expression levels. The estimates remove the technical artifacts, including the location batch effects and the cell-spcific global effects, but retain the biological features for each cell type. Moreover, the cell type effects can be obtained by the `celltype_effects` function. Notice that the first cell type is taken as the baseline cell type implying all zeros in the first column of `celltype_effects_est`.

```
  celltype_mean_expression_est <- celltype_mean_expression(BUSseqfits_example)

## The output format is a matrix.

## Each row represents a gene, and each column corresponds to a cell type.

  head(celltype_mean_expression_est)

##          [,1]     [,2]     [,3]     [,4]
## [1,] 16.11413 2.440190 2.617576 4.930892
## [2,] 14.81980 2.483612 2.590744 4.020473
## [3,] 16.27012 2.607022 2.341530 3.950671
## [4,] 15.68116 2.555482 2.534267 4.433973
## [5,] 17.02854 2.647680 2.731641 4.277442
## [6,] 15.66405 2.543993 2.667953 4.542570

  celltype_effects_est <- celltype_effects(BUSseqfits_res)

## The output format is a matrix.
##
## Each row represents a gene, and each column corresponds to a cell type.

  head(celltype_effects_est)

##      [,1]      [,2]      [,3]      [,4]
## [1,]    0 -2.090990 -2.004485 -1.931996
```

```
## [2,]    0 -1.986020 -1.929446 -2.006292
## [3,]    0 -1.971295 -2.059772 -2.027618
## [4,]    0 -1.997561 -1.982282 -2.008741
## [5,]    0 -1.970303 -1.940841 -2.048707
## [6,]    0 -2.048176 -2.002394 -2.076407
```

5 Intrinsic Gene Identification

```
#obtain the intrinsic gene indicators
intrinsic_gene_indices <- intrinsic_genes_BUSseq(BUSseqfits_res)

#The estimated FDR, the first 500 genes are known intrinsic
#genes in the simulation setting.
false_discovery_ind <- !(intrinsic_gene_indices %in% 1:650)
fdr_est <- sum(false_discovery_ind)/length(intrinsic_gene_indices)
fdr_est
```

```
## [1] 0.003067485
```

Therefore, the true FDR is 0.0030675 less than the estimated FDR, 0.05.

6 Corrected Read Count Data and Visualization

The function `BUSseq_MCMC` not only conducts MCMC sampling and posterior inference, but also imputes the missing data caused by dropout events and corrects batch effects. The function `corrected_read_counts` calculates the corrected read count data of a `BUSseqfits` object. The message is a reminder of the output format, and the output is a `CountData` object. The `summary` command shows the number of batches, genes and cells in each batch.

```
corrected_countdata <- corrected_read_counts(BUSseqfits_res)
```

```
##    correcting read counts...
```

```
##    Correcting read counts takes:  0.029 mins
```

```
## The output format is a "CountData" object with length equal to the batch number.
## Each element of the object is the corrected read count matrix.
## In each matrix, each row represents a gene and each column correspods to a cell.
  class(corrected_countdata)
## [1] "CountData"
  summary(corrected_countdata)
## There are 3 batches and 1000 genes.
## Batch 1 contains 150 cells.
## Batch 2 contains 150 cells.
## Batch 3 contains 150 cells.
##
```

Subsequently, we can compare the raw count data that suffer from batch effects and dropout events, the inferred true expression levels after imputing dropout events, and the corrected count data which remove the batch effects and impute the dropout events. The function `raw_read_counts` and `imputed_read_counts` gives the raw and imputed read count data of a `BUSseqfits` object, respectively. The outputs of them are also a `CountData` object.

```
raw_countdata <- raw_read_counts(BUSseqfits_res)
## The output format is a "CountData" object with length equal to the batch number.
## Each element of the object is the raw read count matrix.
## In each matrix, each row represents a gene and each column correspods to a cell.
imputed_countdata <- imputed_read_counts(BUSseqfits_res)
## The output format is a "CountData" object with length equal to the batch number.
## Each element of the object is the imputed read count matrix.
## In each matrix, each row represents a gene and each column correspods to a cell.
```

The function `heatmap_data_BUSseq` plots the heatmap for the count data across batches for a `CountData` object. The images are saved in the local folder according to the argument `image_dir`. The image name can be modified by the argument `project_name`. The user can specify the argument `gene_ind_set`, which is a vector of gene indices, selecting the genes to be displayed in the heatmap.

```
  #generate the heatmap of raw read count data
  heatmap_data_BUSseq(raw_countdata,
                          project_name = "BUSseq_raw_allgenes",
                          image_dir = "./heatmap")

## pdf
##   2

  #display only the first 100 genes in the heatmap
  heatmap_data_BUSseq(raw_countdata,
                          gene_set = 1:100,
                          project_name = "BUSseq_raw_100genes",
                          image_dir = "./heatmap")

## pdf
##   2
```

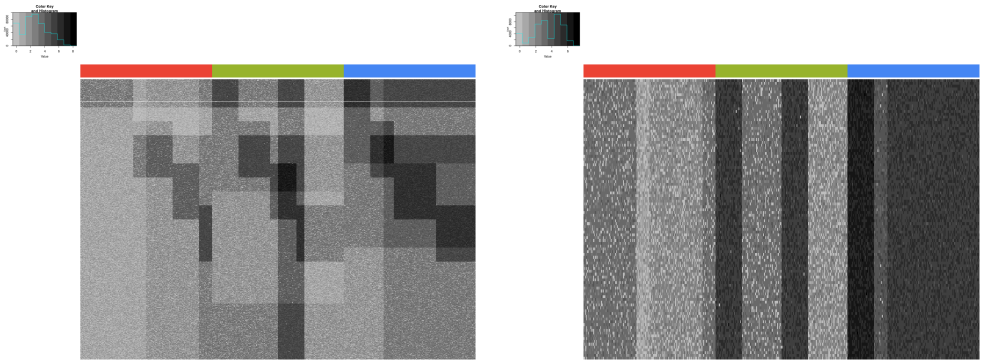


Figure 1: The heatmap of the raw count data of all genes and the first 100 genes

```
  #generate the heatmap of imputed read count data
  heatmap_data_BUSseq(imputed_countdata,
                          project_name = "BUSseq_imputed_allgenes",
                          image_dir = "./heatmap")
```

```
## pdf
##   2
```

```
  #generate the heatmap of corrected read count data
  heatmap_data_BUSseq(corrected_countdata,
                          project_name = "BUSseq_corrected_allgenes",
                          image_dir = "./heatmap")
```

```
## pdf
##   2
```

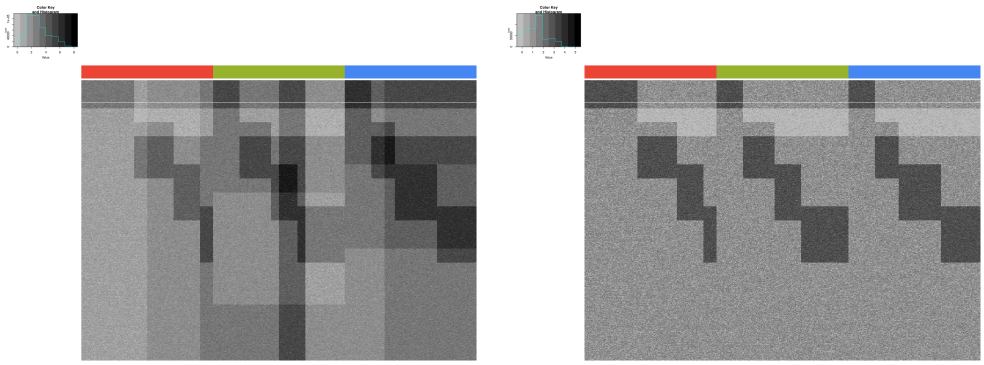


Figure 2: The heatmap for the imputed and corrected count data of all genes

In all these heatmaps, the top bar indicates the batch origin for each cell. Cells under the same color are from the same batch. The batch effects present in the raw data are correctly removed in the corrected count data, and only the biological variabilities are kept. We can also only display the identified intrinsic genes in the corrected count data by setting the argument `gene_set` as `intrinsic_gene_indices`.

```
#Only show the identified intrinsic genes
heatmap_data_BUSseq(corrected_countdata,
                        gene_set = intrinsic_gene_indices,
                        project_name = "BUSseq_corrected_intrinsic_genes",
                        image_dir = "./heatmap")
```

```
## pdf
##   2
```

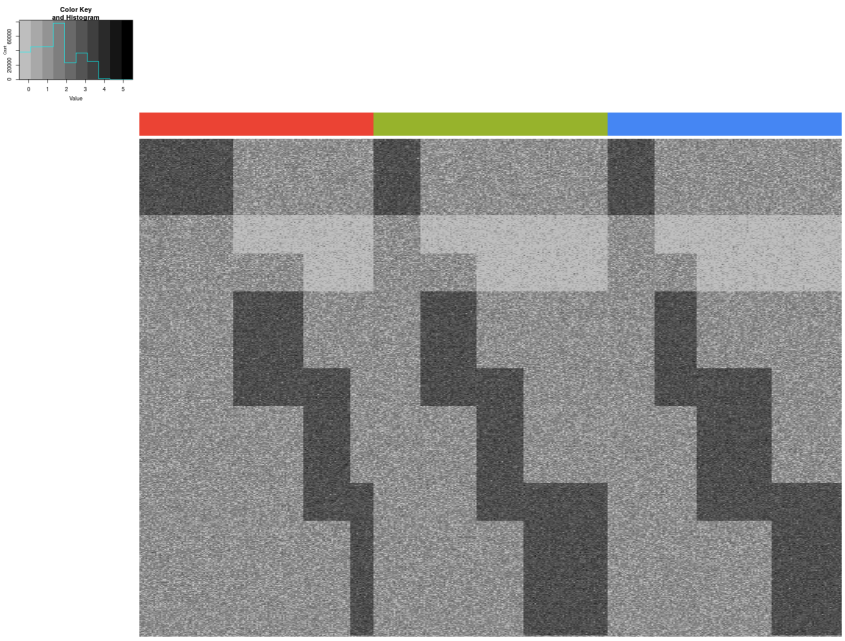


Figure 3: The heatmap for the corrected count data of identified intrinsic genes

7 Model Selection using BIC

If we have no prior knowledge about the cell type number, we can vary the argument `n.celltypes` in the function `BUSseq_MCMC`, e.g., from 2 to 10 and identify the underlying true cell type number K as the one that achieves the minimal BIC.

The user can obtain the BIC value from the `BUSseqfits_res` by the `BIC_BUSseq`.

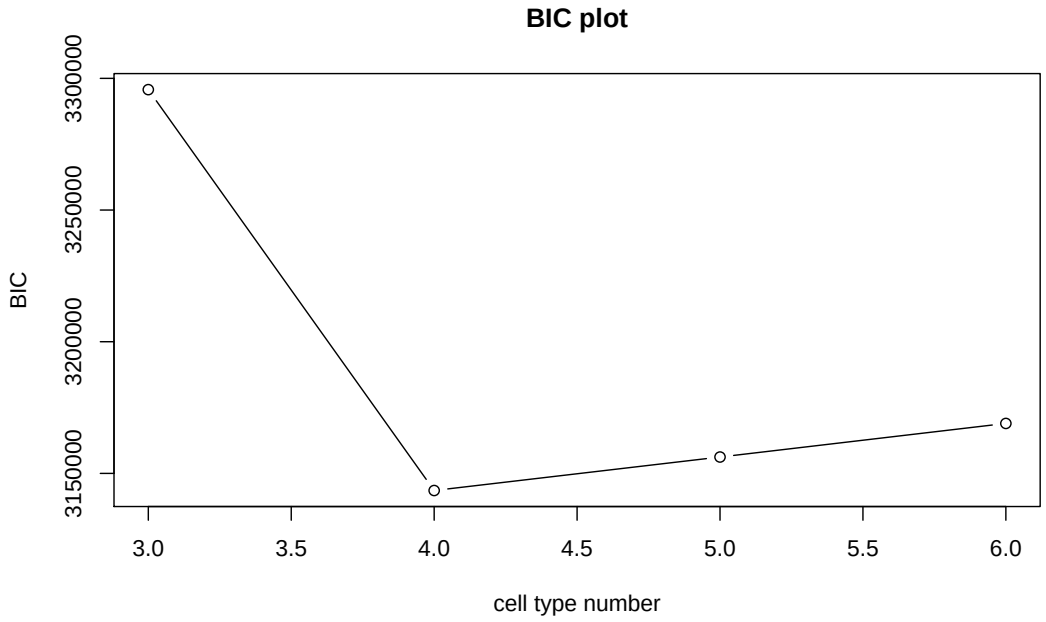
```
   BIC_val <- BIC_BUSseq(BUSseqfits_res)

## BIC is 3143518.89387262

## The output is a scalar.
```

In this example, the underlying true number of cell types is four. For an illustration, we vary the `n.celltypes` from 3 to 6.

```
set.seed(123)
BIC_values <- rep(NA,4)
BUSseqfits_temp <- NULL
for(k in 3:6){
 BUSseqfits_temp[[k-2]] <- BUSseq_MCMC(ObservedData = CountData, n.celltypes = k, n.cores = 4
                                       n.iterations = 500, working_dir = paste0("./K",k))
 BIC_values[k-2] <- BIC_BUSseq(BUSseqfits_temp[[k-2]])
}
```

```
##    conducting the posterior sampling...
##    The MCMC sampling takes:  1.277 mins
##    conducting the posterior inferences...
##    calculating posterior means and posterior takes:  0.029 mins
## BIC is 3295725.29346166
## The output is a scalar.
##    conducting the posterior sampling...
##    The MCMC sampling takes:  1.334 mins
##    conducting the posterior inferences...
##    calculating posterior means and posterior takes:  0.039 mins
## BIC is 3143507.68954213
## The output is a scalar.
##    conducting the posterior sampling...
##    The MCMC sampling takes:  1.381 mins
##    conducting the posterior inferences...
##    calculating posterior means and posterior takes:  0.048 mins
## BIC is 3156220.82641664
## The output is a scalar.
##    conducting the posterior sampling...
##    The MCMC sampling takes:  1.379 mins
```

```
##    conducting the posterior inferences...
##    calculating posterior means and posterior takes:  0.06 mins
## BIC is 3168973.51574592
## The output is a scalar.
plot(3:6, BIC_values, xlab="cell type number", ylab="BIC", main="BIC plot", type="b")
```



The BIC attains the minimum at `n.celltypes`=4, thus correctly recovering the true cell type number.

References

[1] Rhonda Bacher and Christina Kendziorski. Design and computational analysis of single-cell rna-sequencing experiments. *Genome Biology*, 17(1):63, 2016.

[2] W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

[3] Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genetics*, 3(9):e161, 2007.

[4] Jeffrey T Leek. svaseq: removing batch effects and other unwanted noise from sequencing data. *Nucleic Acids Research*, page gku864, 2014.

[5] Fangda Song, Ga Ming Chan, and Yingying Wei. Flexible experimental designs for valid single-cell rna-sequencing experiments allowing batch effects correction. *Manuscript*.