Image Analysis with *beadarray*

Mike Smith

April 27, 2020

Introduction

From version 2.0 *beadarray* provides more flexibility in the processing of array images and the extraction of bead intensities than its predecessor. In the past intensity extraction from array images by *beadarray* attmepted to emulated that performed by Illumina, with minimal opportunities for deviation from this. Whilst the default approach taken in *beadarray* is still to emulated Illumina, we have made each step modular, in order to allow greater flexibility for the user. This vignette is designed to show how one can read the TIFF images from the BeadArray scanner and implement alternative feature intensity extraction algorithms.

1 Reading bead-level data into beadarray

1.1 Standard Illumina Image Processing

The first step in a pipeline for image processing is to read both the TIFF image and the bead-level text file. The text file contains the identities of each bead, as well as the bead-centre coordinates. The image processing methods contained within *beadarray* use these coordinates as seed points for intensity extraction, although one can conceive of approaches where bead-centres are calculated seperately, prior to intensity extraction. However, even with such an approach the Probe ID for each bead will need to be extracted from the *.txt* file.

```
tiff <- readTIFF();
data <- readBeadLevelTextFile();
```

The standard method employed by Illumina's scanner for calculating bead intensity is a four step process, described in Kuhn *et al* [**?**]. It can be summarized as:

- Calculate background value

- Sharpen image
- Calculate foreground value
- Subtract background for foreground to give final intensity

If the function `readIllumina()` is called with `useImages = TRUE` then intensities are extracted using code that gives a very close emulation of that used by Illumina. If one wished to perform this calcuation themselves (outside of `readIllumina()`), it can be done using the following code.

```
bg <- illuminaBackground(tiff, data[,3:4]);
tiffSharp <- illuminaSharpen(tiff);
fg <- illuminaForeground(tiffSharp, data[,3:4]);
finalIntensity <- fg - bg;
```

Each of the functions above take a matrix representing the pixel values from the TIFF image as their first argument. The background and foreground algorithms additionally take a two column matrix containing the coordinates of the bead centres. If one wished to calculate intensities for only a subset of the beads then supplying only the appropriate bead-centres in this step would achieve this.

After calculating intensties they need to be inserted into a *beadLevelData* object. The code below shows how to create a new object and insert intensity values. However, this approach creates an empty *beadLevelData* object, which will be lacking any information except that which the user manually inserts.

```
BLData <- new(Class = "beadLevelData");
BLData <- insertBeadData(BLData, array = 1, what = "Grn", data = finalIntensity)
```

An easier alternative to creating your own *beadLevelData* object is to use the function `readIllumina()` to read the data as described in the main vignette. This ensures that any available data (such as sample IDs, scanner metrics, grid sizes etc.) are read in and stored. One can then choose to overwrite the values generated by `readIllumina()`, or store alternative intensities alongside them.

The example below first reads the data using the standard arguments to `readIllumina()`, which will extract the intensities from the *.txt* file. The second step overwrites those intensities with those we calculated previously (which should be very similar). The final command creates a new entry in the *beadLevelData* object (refered to as 'GrnLog'), that stores the log transform of the values we calculated earlier. In this way the user can store a variety of intensity values if they wish to experiment with alternative forms of background subtraction, gradient removal etc.

```
BLData <- readIllumina();
BLData <- insertBeadData(BLData, array = 1, what = "Grn", data = finalIntensity)
BLData <- insertBeadData(BLData, array = 1, what = "GrnLog", data = log2(finalIntensity))
```

1.2 Alternative Methods

The examples above have focused on applying the same intensity extraction algorithms that are employed by Illumina. However, one may wish to employ an alternative algorithm to test its performance. The example below implements an alternative method of calculating the background intensity values, as recommended by Smith *et al* [**?**].

```
bg <- medianBackground(tiff, data[,3:4]);
```

We can then use the new background intensities in the same way as previously, before inserting them into the *beadLevelData* object.

2 Parallel Processing

We have included some support for parallel processing in the functions to perform sharpening of the image and the two background calculation methods. These can offer some increase in throughput when one is using a single computer to analyse a small number of samples. However if one is dealing with a large number of arrays then there are probably more efficient mechanisms to achieve speedup, such as reading seperate chips on multiple machines (or `R` sessions) and combining the data after they have been read.

This multicore support is implemented at the `C` level using the **OpenMP** library. Unfortunately adding support for this generates a warning on Bioconductor, so support needs to be added manually and the package build from source. The procedure is slightly different for users on Linux and Windows machines.

Linux users should create a file called `Makevars` in the `beadarray/src` directory and add the following two lines before building the package from source.

```
PKG_CFLAGS=-fopenmp
PKG_LIBS=-lgomp
```

Windows users should create a file called `Makevars.win` in the `beadarray/src` directory and add the following two lines before building the package from source.

```
PKG_CFLAGS=-fopenmp
PKG_LIBS=-lgomp -mthreads -lpthreadGC2
```

3 Session Info

```
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server 2012 R2 x64 (build 9600)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets
## [7] methods   base
##
## other attached packages:
## [1] beadarray_2.38.0    hexbin_1.28.1       Biobase_2.48.0
## [4] BiocGenerics_0.34.0 knitr_1.28
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.0.0       xfun_0.13
##  [3] purrr_0.3.4            reshape2_1.4.4
##  [5] lattice_0.20-41        colorspace_1.4-1
##  [7] vctrs_0.2.4            htmltools_0.4.0
##  [9] stats4_4.0.0           yaml_2.2.1
## [11] BeadDataPackR_1.40.0   blob_1.2.1
## [13] rlang_0.4.5            pillar_1.4.3
## [15] glue_1.4.0             DBI_1.1.0
## [17] bit64_0.9-7            GenomeInfoDbData_1.2.3
## [19] lifecycle_0.2.0        plyr_1.8.6
## [21] stringr_1.4.0          zlibbioc_1.34.0
## [23] munsell_0.5.0          gtable_0.3.0
## [25] memoise_1.1.0          evaluate_0.14
```

```
## [27] IRanges_2.22.0          GenomeInfoDb_1.24.0
## [29] AnnotationDbi_1.50.0    highr_0.8
## [31] illuminaio_0.30.0       Rcpp_1.0.4.6
## [33] openssl_1.4.1           scales_1.1.0
## [35] BiocManager_1.30.10     limma_3.44.0
## [37] base64_2.0              S4Vectors_0.26.0
## [39] XVector_0.28.0          bit_1.1-15.2
## [41] askpass_1.1             BiocStyle_2.16.0
## [43] ggplot2_3.3.0           digest_0.6.25
## [45] stringi_1.4.6           dplyr_0.8.5
## [47] GenomicRanges_1.40.0    grid_4.0.0
## [49] tools_4.0.0             bitops_1.0-6
## [51] magrittr_1.5            RCurl_1.98-1.2
## [53] RSQLite_2.2.0           tibble_3.0.1
## [55] crayon_1.3.4            pkgconfig_2.0.3
## [57] ellipsis_0.3.0          assertthat_0.2.1
## [59] rmarkdown_2.1           R6_2.4.1
## [61] compiler_4.0.0
```