Total copy-number segmentation using CBS

Henrik Bengtsson

NA

Abstract

The Circular Binary Segmentation (CBS) method partitions a genome into segments of constant total copy numbers (TCNs) based on DNA microarray data. The method also calls .... CBS was designed to work with data from any DNA microarray technology and generation, including Affymetrix and Illumina.

This document shows how to use the *PSCBS* package to run CBS on a tumor sample.

Keywords: copy numbers, genomic aberrations

This vignette is distributed as part of the PSCBS package, which is available on CRAN (`https://cran.r-project.org/`). The authors very much appreciate feedback on this document.

Contents

1 Background

We will here use a small example data set to illustrate how to setup the data in a format suitable for CBS, how to identify segments, how to call them, and how to plot and export the segmentation results. The statistical model and the algorithm behind CBS is explained in detail in Olshen *et al.* (2004); Venkatraman and Olshen (2007).

2 Preparing data to be segmented

The CBS method requires total copy-number (TCN) estimates. More precisely, it requires TCN ratios for a sample of interest relative to a reference (y). The genomic location of the loci in form of chromosome and physical position are also required.

2.1 Locus-level total copy-number signals

In this example we will use a small example data set part of the *PSCBS* package. It can be loaded as:

```
> data <- PSCBS::exampleData("paired.chr01")
> data <- data[, c("chromosome", "x", "CT")]
> colnames(data)[3] <- "y"
> str(data)
'data.frame': 73346 obs. of  3 variables:
 $ chromosome: int  1 1 1 1 1 1 1 1 1 1 ...
 $ x         : int  1145994 2224111 2319424 2543484 2926730 2941694 3084986 3155127..
 $ y         : num  1.625 1.071 1.406 1.18 0.856 ...
```

In additional to the mandatory fields (`chromosome`, `x`, and `C` this data set also contains .... The latter will not be used here.

2.2 Dropping TCN outliers

There may be some outliers among the TCNs. In CBS (Olshen *et al.*, 2004; Venkatraman and Olshen, 2007), the authors propose a method for identifying outliers and then to shrink such values toward their neighbors (”smooth”) before performing segmentation. At the time CBS was developed it made sense to not just to drop outliers because the resolution was low and every datapoint was valuable. With modern technologies the resolution is much higher and we can afford dropping such outliers, which can be done by:

```
> data <- dropSegmentationOutliers(data)
```

Dropping TCN outliers is optional.

3 CBS segmentation

3.1 Skipping centromeres and other large gaps

The CBS method does not take the physical locations (in units of nucleotides) of the loci in to account when segmenting the data, only their relative ordering along the genome. This means that after having ordered the loci along genome, it will treat two ”neighboring” loci that are on both sides of the centromere equally as two neighboring loci that are only few hundred bases apart. This may introduce erroneous change points that appears to be inside the centromere and biological impossible interpretation of the identified copy-number states. The same issues occur for other large gaps of the genome where there are no observed signals.

To avoid this, although not mandatory, we will locate all gaps of the genome where there are no observed loci. As a threshold we will consider a region to be a "gap" if the distance between the two closest loci is greater than 1Mb.

```
> gaps <- findLargeGaps(data, minLength = 1e+06)
> gaps
  chromosome     start       end   length
1          1 120992604 141510002 20517398
```

which shows that there is a 20.5Mb long gap between 121.0Mb and 141.5Mb on Chromosome 1. This is the centromere of Chromosome 1. Gaps cannot be specified directly. Instead they need to be give as part of a set of "known" segments, which is done as:

```
> knownSegments <- gapsToSegments(gaps)
> knownSegments
  chromosome    start      end   length
1          1     -Inf 1.21e+08      Inf
2          1 1.21e+08 1.42e+08 20517398
3          1 1.42e+08      Inf      Inf
```

Below, we will use this to tell CBS to segment Chromosome 1 in three independent segments, where the first segments is from the beginning of the chromosomes (hence '-Inf') to 120.1Mb, the second from 120.1-141.5Mb (the above gap), and the third is from 141.5Mb to the end of the chromosome (hence '+Inf'). Just as CBS segments chromosomes independently of each other, it also segments priorly known segments independently of each other. Specifying known segments is optional.

3.2 Identifying TCN segments

We are now ready to segment the locus-level TCN signals. This is done by[1]:

```
> fit <- segmentByCBS(data, knownSegments = knownSegments,
+     seed = 48879, verbose = -10)
```

Note that this may take several minutes when applied to whole-genome data.

The result of the segmentation is a set of segments identified to have the same underlying TCN levels. In this particular case, 9 TCN segments were found:

```
> getSegments(fit, simplify = TRUE)
  sampleName chromosome     start       end nbrOfLoci mean
1       <NA>          1    554484  86993745     26846 1.38
2       <NA>          1  86993745  87005243         2 3.19
3       <NA>          1  87005243 120992603     10647 1.39
4       <NA>          1 120992604 141510002         0   NA
5       <NA>          1 141510003 185527989     13434 2.07
6       <NA>          1 185527989 199122065      4018 2.71
7       <NA>          1 199122065 206512702      2755 2.59
8       <NA>          1 206512702 206521352        14 3.87
9       <NA>          1 206521352 247165315     15581 2.64
```

Note how Segment #4 has no mean-level estimates. It is because it corresponds to the centromere (the gap) that was identified above. CBS did indeed try to segment it, but since there are no data points, all estimates are missing values.

[1]We fix the random seed in order for the results of this vignette to be exactly reproducible.

3.3 Displaying genomic TCN profiles

To plot the TCN segmentation results, do:

```
plotTracks(fit)
```

which displays TCN as in Figure 1. To zoom in on a particular region, do:

```
plotTracks(fit, xlim=c(120,244)*1e6)
```

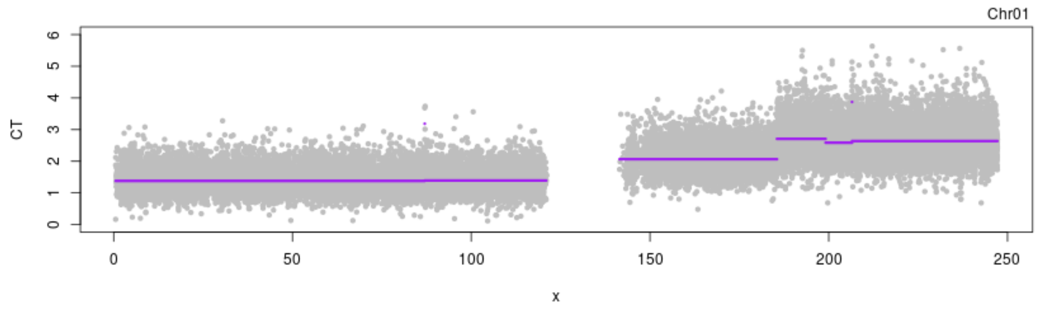

Figure 1: Segments identified by CBS. The TCN signals with the TCN mean levels (purple).

4 Calling segments

TBA.

5 Saving results

5.1 Writing segments to a tab-delimited text file

To write the TCN segmentation results to file, do:

```
pathname <- writeSegments(fit, name="MySample", simplify=TRUE)
```

6 Experimental

In this section we illustrate some of the ongoing and future work of the PSCBS package. Please be aware that these methods are very much under construction, possibly incomplete and in worst case even incorrect.

6.1 Pruning segmentation profile

By using hierarchical cluster of the segment means it is possible to prune the TCN profile such that change points with very small absolute changes are dropped. If change points are dropped this way, this results in a smaller number of segments, which are hence longer.

```
> fitP <- pruneByHClust(fit, h = 0.25, verbose = -10)
```

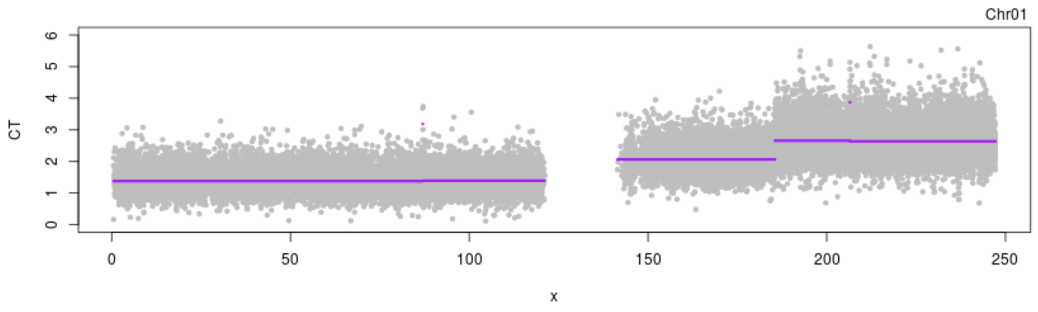

Figure 2: Pruned TCN segments plotted as in Figure 1.

6.2 Report generation

A multipage PDF report that contains both whole-genome and per-chromosome summaries and figures can be generated by:

```
> report(fit, sampleName="CBS", studyName="CBS-Ex", verbose=-10)
```

By default, the reports are written to directory `reports/<studyName>/` under the current working directory. In addition to the PDF, that directory also contains subdirectory `figures/` holding all generated figure files (e.g. PNGs and PDFs) for easy inclusion elsewhere.

References

Olshen, A. B., Venkatraman, E. S., Lucito, R., and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based dna copy number data. *Biostatistics*, **5**(4), 557–572.

Venkatraman, E. S. and Olshen, A. B. (2007). A faster circular binary segmentation algorithm for the analysis of array CGH data. *Bioinformatics*, **23**(6), 657–663.

Appendix

Session information

- R version 4.1.1 Patched (2021-10-20 r81080), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.6 LTS`
- Random number generation:
- RNG: `L'Ecuyer-CMRG`
- Normal: `Inversion`
- Sample: `Rejection`
- Matrix products: default
- BLAS: `/home/hb/software/R-devel/R-4-1-branch/lib/R/lib/libRblas.so`
- LAPACK: `/home/hb/software/R-devel/R-4-1-branch/lib/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: DNAcopy 1.66.0, PSCBS 0.66.0, R.devices 2.17.0-9000, R.methodsS3 1.8.1-9001, R.oo 1.24.0, R.utils 2.11.0
- Loaded via a namespace (and not attached): Cairo 1.5-12.2, R.cache 0.15.0, R.rsp 0.44.0-9000, aroma.light 3.22.0, base64enc 0.1-3, codetools 0.2-18, compiler 4.1.1, digest 0.6.28, future 1.22.1-9004, globals 0.14.0-9000, listenv 0.8.0-9000, matrixStats 0.61.0, parallel 4.1.1, parallelly 1.28.1-9000, tools 4.1.1

This report was automatically generated using `rfile()` of the R.rsp package. Total processing time after RSP-to-R translation was 13.44 secs.