Processing and Visualisation of High-Throughput Sequencing Data with `ShortRead` and `HilbertCurveDisplay`

Simon Anders

European Bioinformatics Institute,
Hinxton, Cambridge, UK

`sanders@fs.tum.de`

2008-07-21

Abstract

This document serves a double purpose: (i) It explains the use of the Bioconductor packages *HilbertVis* and *HilbertVisGUI*. This pair of packages offers a tool to visualise very long one-dimensional data vectors (with up to billions of entries) in an efficient fashion that allows to get a quick impression of the spatial distribution and rough shape of the features present in the data. This is especially useful in the initial exploration of high-resolution position-dependent genomic data, such as tiling array or ChIP-Seq data. (ii) It provides a specific example by walking the reader through the task of processing ChIP-Seq data using the stand-alone alignment tool Maq and the Bioconductor packages Biostrings, ShortRead and HilbertVis/HilbertVisGUI.

Note: If you are only interested in the use of the `HilbertVis`/`HilbertVisGUI` packages, you can skip the first section and start reading at Section 6.

Note: If you have trouble installing the package `HilbertVisGUI`, read the file `INSTALL` in the package.

1   Introduction

Bioconductor offers substantial support for genomic experiments, which, for the case of microarray platforms, including tiling arrays, has reached maturity already a while ago. For data from high-throughput sequencing experiments, development of new tools is currently (mid 2008) ongoing. In this document, I would like to show what can already been done by re-doing step for step the analysis of an already published Solexa ChIP-Seq experiment.

I use this to demonstrate some aspects of the ShortRead package (by M. Morgan, [Mor]) and the use of my packages "HilbertVis" and "HilbertVisGUI". ShortRead introduces data structures to represent aligned short sequence reads and offers functions to read in such data from files output by the SolexaPipeline (the software that Illumina provides with its GenomeAnalyzer machine) or by Maq (a stand-alone alignment program, [Li]). ShortRead's data structures are based on the infrastructure provided by Biostrings.

As an example, we use data from a published study, Ref. [BCC⁺07], on histone methylations in the human genome. Although this was not the main focus of that study, we re-analyse the data for histone methylation patterns H3K4me1 and H3K4me3, as these are data sets of manageable size. We first re-do the alignment with Maq, then use ShortRead to read the result into R and then visualise the data with HilbertVis.

2 The example data

The authors of our example have deposited their raw data in the NCBI's Provisional Short Read Archive (`http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi`) under accession number SRA000206. In the provisional archive, you can access an FTP server by clicking on "SRA FTP" and will find there a directory named "SRA000206" that contains all the data as output by Bustard, the base-calling program in the SolexaPipeline, as well as a number of fairly self-explanatory XML files with meta data. There are 3 lanes for H3K4me1 and 7 lanes for H3K4me3.

I have used Maq to align the reads from these lanes against the human reference genome as provided on Ensembl. Doing so requires converting the `_seq.txt` and `_prb.txt` files for the lanes to the Sanger Institute's FASTQ format and on to Maq's BFQ (binary FASTQ format). Likewise, the reference genome is converted to one large BFA (binary FASTA) file. Then, the `maq map` command may be used to perform the alignment. As these steps are described in the documentation on the Maq web site, I do not go into detail here.[1] In the end, we have, as output from Maq, a mapping file for each lane. I have put these files onto my web page. So, if you want to try out the following steps for yourself, please download them from `http://www.ebi.ac.uk/~anders/ShortReadExampleData/`. Note, however, that you should use a machine with at least 4 GB of RAM to perform the examples.

Of course, Maq is not the only choice to align the reads to the genome. You may as well use Eland (the alignment program that comes with the SolexaPipeline), which can be read in as well by ShortRead, so that the following steps apply to this case as well. Within certain limits, the matching functionality of the Biostrings package allows you to even do everything within R. Finally, there are other alignment tools specialised for high-throughput sequences (e.g., Soap, rmap, shrimp) whose output cannot yet be read by ShortRead but who may be supported in the future.

3 Reading in the alignment

Assume that the current working directory contains two sub-directories, names `H3K4me1` and `H3K4me3`. Then we can read in all the files of pattern `runxxlanex.map` with the following commands:

```
> library("ShortRead")
> maps.me1 <- sapply( list.files( "H3K4me1", "run.*lane.\\.map" ),
+    function(filename) readAligned( "H3K4me1", filename, type="MAQMap" ) )
> maps.me3 <- sapply( list.files( "H3K4me3", "run.*lane.\\.map" ),
+    function(filename) readAligned( "H3K4me3", filename, type="MAQMap" ) )
```

Here, `readAligned` takes three arguments: the directory that contains the map file, the name of the map file, and the type of data to be read, `MAQMap`. You may also use the type `SolexaExport` to read in mappings produced by Eland (see help page for `readAligned` for details). In any case, the function `readAligned` returns an S4 object of class `AlignedRead`.

---

[1] However, feel free to contact me if you want to know details.

4 The class `AlignedRead`

An `AlignedRead` object is conceptionally quite similar to a data frame. It contains as many "rows" as there are mapped reads:

```
> length( maps.me1$run4_lane8.map )
```

```
[1] 3465080
```

For each read, all the data parsed from the map file are stored. Think of these types of data as of columns of a data frame, even though you do not access them with the `$` operator but with accessor functions. The "columns" `chromosome`, `position` and `width` show where in the genome the reads were mapped:

```
> head( chromosome( maps.me1$run4_lane8.map ) )
```

```
[1] 10 10 10 10 10 10
113 Levels: 10 11 12 13 14 15 16 17 18 19 1 20 21 22 2 3 4 5 6 7 8 9 MT X ... NT_113898
```

```
> head( position( maps.me1$run4_lane8.map ) )
```

```
[1] 50547 53424 58681 61890 64043 66900
```

```
> head( width( maps.me1$run4_lane8.map ) )
```

```
[1] 25
```

As we see, the first 6 of the 3.4 mio reads in lane 8 of run 4 were all mapped to chromosome 10, to the given positions, and extending from there all by 25 bp.[2]

The actual reads are stored as well,

```
> head( sread( maps.me1$run4_lane8.map ) )
```

```
  A DNAStringSet instance of length 6
    width seq
[1]    25 CCAGGAGAATATGCAATGATGACAA
[2]    25 TATAGAGCATTAAACCACCAAAGCT
[3]    25 TAACCAACTCAAGTGCCCATCAGTG
[4]    25 TGATTGTGCCACTGCACTCTAGCAA
[5]    25 TTGCTGGCACCAGGGACCAGGAGGA
[6]    25 TACCATCTCACACCACTTAGAATGG
```

as are the reads' identifiers (which here encode their position on the lane):

```
> head( id( maps.me1$run4_lane8.map ) )
```

```
  A BStringSet instance of length 6
    width seq
[1]    14 R:8:34:810:204
[2]    14 R:8:82:891:530
[3]    15 R:8:136:501:225
[4]    13 R:8:68:11:564
[5]    15 R:8:101:523:977
[6]    14 R:8:15:487:873
```

---

[2]Note that Maq stores the aligned reads in order of their alignment. Hence, we start with very low base-pair indices, which then increase. Maq has also started with chromosome 10, as that one happened to be the first one in the BFA file.

These last two objects are not ordinary R `character` vectors but `DNAStringSet` and `BStringSet` objects. These are specialised data structures provided by the `Biostrings` package designed to handle large amounts of character (or sequence) data. They are not elementary-type vectors but S4 objects. (See the Biostrings vignette for details.) As they only mimic a vector they cannot be columns of a data frame. This is the reason why `AlignedRead` is not a data frame although its structure is reminiscent of one.

Other information stored in the `AlignedRead` object is the base-call quality as reported by Bustard, here given in FASTQ quality string representation. (See the Maq web site for an explanation of the format.)

```
> head( quality( maps.me1$run4_lane8.map ) )

class: FastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    25 +//59>@II1ACIIFIIII:IIIII
[2]    25 IIIIIIIIIIIIIII7II5CI<.7+&
[3]    25 III1IIII4$@IIFI%3/DI$%A8*+
[4]    25 +&%,,;+@;*7%6I+;*I>+%,@I@I
[5]    25 +IIIIII5HB3IHIE1&+++8,7+)
[6]    25 II%5I9.&II<I%8'%#7*,0%@9$
```

Maq calculates from this information and from the uniqueness and perfectness of the alignment an alignment score, which is stored in an `alignQuality` object:

```
> alignQuality( maps.me1$run4_lane8.map )

class: IntegerQuality
quality: 0 0 ... 9 9 (3465080 total)
```

The actual integer vector of qualities (one number per read) can be obtained with the accessor function `quality`

```
> head( quality( alignQuality( maps.me1$run4_lane8.map ) ) )

[1] 0 0 0 0 0 0
```

The accessor function `strand` reports whether the read was mapped to the "+" or to the "–" strand of the chromosome. It returns a factor with two levels:

```
> head( strand( maps.me1$run4_lane8.map ) )

[1] - + + - + +
Levels: - +
```

Remember that Solexa sequencing is not strand-specific. The information, from which strand the sequence actually was precipitated, got lost during bridge PCR amplification. Hence, it is better to think of this factor not as information on the strand but rather on the *direction* of the fragment Have a look at Fig. 1 for an illustration.

The fields described so far are available for all `AlignedRead` objects. Depending on the alignment software that was used additional information may be available. The slot `alignData` is meant to hold such information
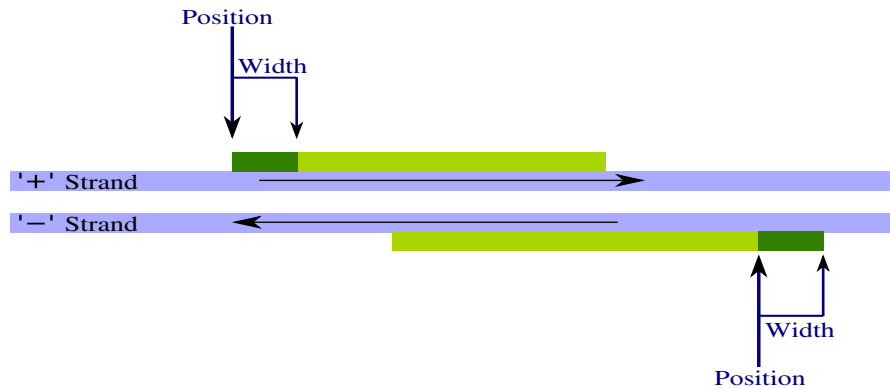
4

Figure 1: The `strand` information shows how the read is aligned against the genome. If `strand` is `"+"`, the `position` indicates the start of the DNA read (dark green) as well as the start of the whole fragment. The part of the fragment that was not read (light green) extends to the right (i.e. towards larger chromosome coordinates). If `strand` is `"-"`, then the fragment extends to the left. As `position` always indicates the left edge of the read (but not necessarily an edge of the whole fragment) it now points to a position within the whole fragment.

```
> alignData( maps.me1$run4_lane8.map )

An object of class "AlignedDataFrame"
  readName: 1, 2, ..., 3465080  (3465080 total)
  varLabels and varMetadata description:
    nMismatchBestHit: Number of mismatches of the best hit
    mismatchQuality: Sum of mismatched base qualities of the best hit
    nExactMatch24: Number of 0-mismatch hits of the first 24 bases
    nOneMismatch24: Number of 1-mismatch hits of the first 24 bases
```

`AlignedDataFrame` is a subclass of `AnnotatedDataFrame`. Hence, we can see the meaning of the columns from the meta information displayed above and access the underlying data frame with `pData`:

```
> head( pData( alignData( maps.me1$run4_lane8.map ) ) )

  nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
1                0               0             3              0
2                0               0             2             85
3                2              17             0              1
4                1               4             1             34
5                1              30             0              4
6                1               4            85             85
```

5 Pile-up

In ChIP-Seq, one is usually interested in the number of precipitated DNA fragments in the sample that were mapped to each genomic locus. This is best represented by what I call a "pile-

up vector[3]. A pile-up vector is a very long `integer` vector with as many elements as there are base pairs in the chromosome under consideration. Each vector element counts the number of fragments that were mapped such that they cover this base pair. The function `pileup` in the `ShortRead` package calculates such a vector from alignment information.

In order to allocate a vector of the right size, `pileup` needs to know the length of the chromosome. `readBfaToc` obtains the lengths of all sequences in a BFA file (binary FASTA, the compressed FASTA format used by Maq). As a BFA file has a table of content at the beginning, `readBfaToc` only has to read the header of the BFA file and is hence quite fast.

```
> seqlens <- readBfaToc( "Homo_sapiens.NCBI36.48.dna.all.bfa" )
> seqlens
        10         11         12         13         14         15         16         17
 135374737  134452384  132349534  114142980  106368585  100338915   88827254   78774742
        18         19          1         20         21         22          2          3
  76117153   63811651  247249719   62435964   46944323   49691432  242951149  199501827
         4          5          6          7          8          9         MT          X
 191273063  180857866  170899992  158821424  146274826  140273252      16571  154913754
         Y  NT_113887  NT_113947  NT_113903  NT_113908  NT_113940  NT_113917  NT_113963
  57772954       3994       4262      12854      13036      19187      19840      24360
 NT_113876  NT_113950  NT_113946  NT_113920  NT_113911  NT_113907  NT_113937  NT_113941
     25994      28709      31181      35155      36148      37175      37443      37498
 NT_113909  NT_113921  NT_113919  NT_113960  NT_113945  NT_113879  NT_113938  NT_113928
     38914      39615      40524      40752      41001      42503      44580      44888
 NT_113906  NT_113904  NT_113873  NT_113966  NT_113943  NT_113914  NT_113948  NT_113886
     46082      50950      51825      68003      81310      90085      92689      96249
 NT_113932  NT_113929  NT_113878  NT_113927  NT_113900  NT_113918  NT_113875  NT_113942
    104388     105485     106433     111864     112804     113275     114056     117663
 NT_113926  NT_113934  NT_113954  NT_113953  NT_113874  NT_113883  NT_113924  NT_113933
    119514     120350     129889     131056     136815     137703     139260     142595
 NT_113884  NT_113890  NT_113870  NT_113881  NT_113939  NT_113956  NT_113951  NT_113902
    143068     143687     145186     146010     147354     150002     152296     153959
 NT_113913  NT_113958  NT_113949  NT_113889  NT_113936  NT_113957  NT_113961  NT_113925
    154740     158069     159169     161147     163628     166452     166566     168820
 NT_113882  NT_113916  NT_113930  NT_113955  NT_113944  NT_113901  NT_113905  NT_113872
    172475     173443     174588     178865     182567     182896     183161     183763
 NT_113952  NT_113912  NT_113935  NT_113880  NT_113931  NT_113923  NT_113915  NT_113885
    184355     185143     185449     185571     186078     186858     187035     189789
 NT_113888  NT_113871  NT_113964  NT_113877  NT_113910  NT_113962  NT_113899  NT_113965
    191469     197748     204131     208942     211638     217385     520332    1005289
 NT_113898
   1305230
```

If you try to reproduce this example, you may not have the BFA file[4]. So, you can obtain the object `seqlens` manually with the following command (which omits the `NT_xxxxx` contigs):

```
> seqlens <- c( `10`=135374737, `11`=134452384, `12`=132349534, `13`=114142980,
+ `14`=106368585, `15`=100338915, `16`=88827254, `17`=78774742, `18`=76117153,
```

---

[3]Maybe, there is another, more established term that I am not aware of. I borrow from the terminology used with Maq, where a related function is called "pileup".

[4]I have not put it on my web page as it is very big and easily created from the Ensembl files.
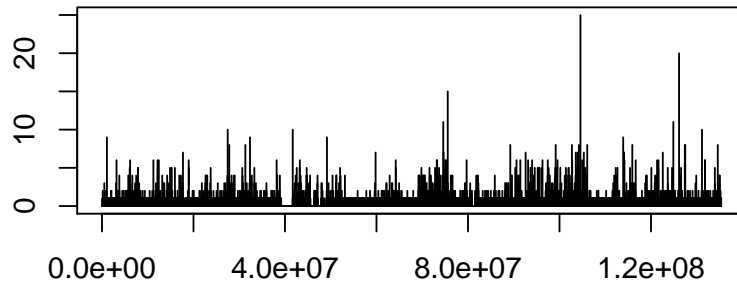
Figure 2: Output of `plotLongVector( me3.p10 )`.

```
+ `19`=63811651, `1`=247249719, `20`=62435964, `21`=46944323, `22`=49691432,
+ `2`=242951149, `3`=199501827, `4`=191273063, `5`=180857866, `6`=170899992,
+ `7`=158821424, `8`=146274826, `9`=140273252, MT=16571, X=154913754, Y=57772954 )
```

In order to get a pileup for, say, chromosome 10, using only mappings in `maps.me3$run13_lane4.map` with a mapping quality of at least 15, we first create a new `AlignedRead` object containing only these reads

```
> foo <- maps.me3$run13_lane4.map[
+     chromosome( maps.me3$run13_lane4.map ) == "10" &
+     quality(alignQuality( maps.me3$run13_lane4.map )) >= 15 ]
```

and then run the `pileup` function on these:

```
> me3.p10 <- pileup( position(foo), width(foo), seqlens[["10"]] )
> str(me3.p10)

 int [1:135374737] 0 0 0 0 0 0 0 0 0 0 ...
```

In principle, we could now plot this vector simply by typing

```
> plot( me3.p10, type='h' )
```

However, this command takes very long, as it plots one needle for each vector element, spending most of its time plotting over and over at the same spot. The function `plotLongVector` (in `HilbertVis`) produces the same plot with a decent speed:

```
> library("HilbertVis")
> plotLongVector( me3.p10 )
```

[Output: See Fig. 2.]

It does so by first partitioning the vector in 4,000 parts of equal length, getting the maximum out of each part, and returning it as a vector, which has now only length 4,000. In case you want

7

to write your own plotting function (because `plotLongVector` is rather rudimentary), you can use the function `shrinkVector` (in `HilbertVis`) to accomplish this.

In the form used above, the function `pileup` counts only which base pairs the actual read covers. Typically, the read length (here: 25 nt) is much shorter than the length of the DNA fragments. In the present data, the length of the fragments after sonication, adaptor ligation and gel-electrophoretic size selection was about 220 bp including adaptors, i.e., approx. 185 bp without adaptors. Given that the immuno-precipitated histone can be anywhere on the fragment, not necessarily within the part at the end that is actually sequenced (the "read") we get a less biased picture by incorporating this information into the calculation of the pile-up vector. As illustrated in Fig. 1 we cannot simply replace the second argument with 185 because depending on the strand value the position my need shifting. The `pileup` function does this calculation correctly when called thus:

```
> me3.p10 <- pileup( position(foo), 185, seqlens[["10"]], strand(foo), width(foo) )
```

The vector `me3.p10` incorporated only the information from one lane. We get better count statistics by getting such a vector for each lane from the H3K4me3 sample and then simply summing them up element-wise:

```
> chr <- "10"
> me3.p10 <- rep.int( 0, seqlens[[chr]] )
> for( mp in maps.me3) {
+    foo <- mp[ chromosome(mp) == chr & quality(alignQuality( mp )) >= 15 ]
+    me3.p10 <- me3.p10 +
+        pileup( position(foo), 185, seqlens[[chr]], strand(foo), width(foo) ) }
```

Note that a `for` loop is used here instead of the use of `sapply` together with `rowSums`. The latter may look more natural in R but as it builds up a two-dimensional array of all the intermediate pile-up vectors it requires too much memory. Even with the `for` loop we give the garbage collector quite some work, and the operation takes a while. Let's do the same for "me1":

```
> me1.p10 <- rep.int( 0, seqlens[[chr]] )
> for( mp in maps.me1) {
+    foo <- mp[ chromosome(mp) == chr & quality(alignQuality( mp )) >= 15 ]
+    me1.p10 <- me1.p10 +
+        pileup( position(foo), 185, seqlens[[chr]], strand(foo), width(foo) ) }
```

As "me1" has only 3 lanes as opposed to "me3"'s 7 lanes, we multiply it with 7/3 to get both pile-ups on the same scale:

```
> me1.p10 <- me1.p10 * 7/3
```

6 Visualisation with Hilbert curve plots

6.1 The Hilbert curve

Note: If you have skipped the previous sections as you only want to read about `HilbertVis`, here is what you need to know in order to start reading here: We have re-analysed part of the ChIP-Seq experiments done in Ref. [BCC+07], namely the data regarding histone methylation patterns H3K4me1 and H3K4me3. We have constructed two very long "pile-up" vectors, `me1.p10` and `me3.p10`, which represent human chromosome 10 and have a length of 135374737, corresponding to the number of base pairs in chromosome 10. Each element corresponds to a base pair and counts how many precipitated and
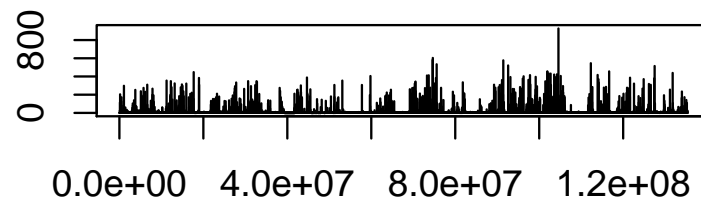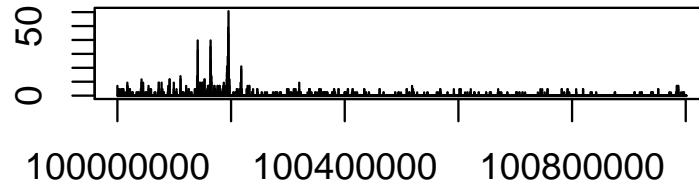
Chr 10, H3K3me1
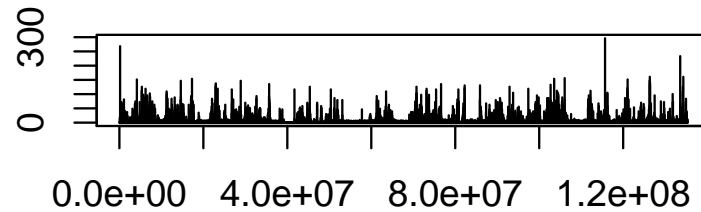
Chr 10, H3K3me3

Figure 3: Pile-up representation of the ChIP-Seq data for H3K4me1 and H3K4me3, depicting the whole of chromosome 10.

sequenced DNA fragments within the respective sample (H3K4me1 or H3K4me3) cover this position. To do your own experiments with these vectors, you may download them as R data file from `http://www.ebi.ac.uk/~anders/ShortReadExampleData/meX.p10.Rdata`.

Note 2: Since I have written this vignette, I have restructured teh package and split it into two parts, called "HilbertVis" and "HilbertVisGUI". This text focuses on the functionality of "HilbertVisGUI", which provides an interactive tool to explore data using teh visualisation technique desribed in the following. "HilbertVis" contains further functions to produce the same kind of images but without interactive tools, i.e. solely from the R command line. If you want to know more about these functions, which are not (yet) mentioned in the present text, see the vignette "Visualising very long data vectors with the Hilbert curve", which is included in the "HilbertVis" package.

A first approach to visualising the two vectors is plotting them with the `plotLongVector` function described above:
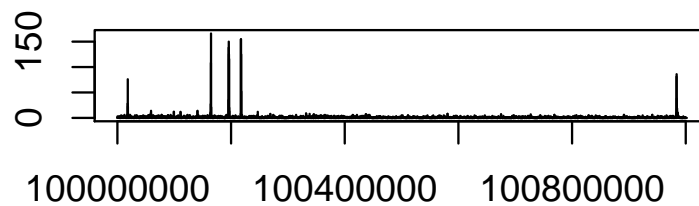
Chr 10, H3K3me1

Chr 10, H3K3me3

Figure 4: Zoom into a small portion of Fig. 3.

```
> library( HilbertVis )
> library( HilbertVisGUI )
> par( mfrow = c(2,1) )
> plotLongVector( me1.p10, main="Chr 10, H3K3me1" )
> plotLongVector( me3.p10, main="Chr 10, H3K3me3" )
```

[Output: Fig. 3.]

The two vectors do look different but it is hard to make out what gives rise to the difference. Is the number of peaks different, or their distribution, or their typical width? Given that each pixel on the x axis corresponds to more than 100 kp, each of the needle can as well be a small peak, only a few fragment lengths wide, a wide peak with a base of tens of kb, or even a cluster of several peaks. We might zoom in somewhere but this is not too illuminating:

```
> par( mfrow = c(2,1) )
> plotLongVector( me1.p10[100000000:101000000], main="Chr 10, H3K3me1", offset=100000000 )
```
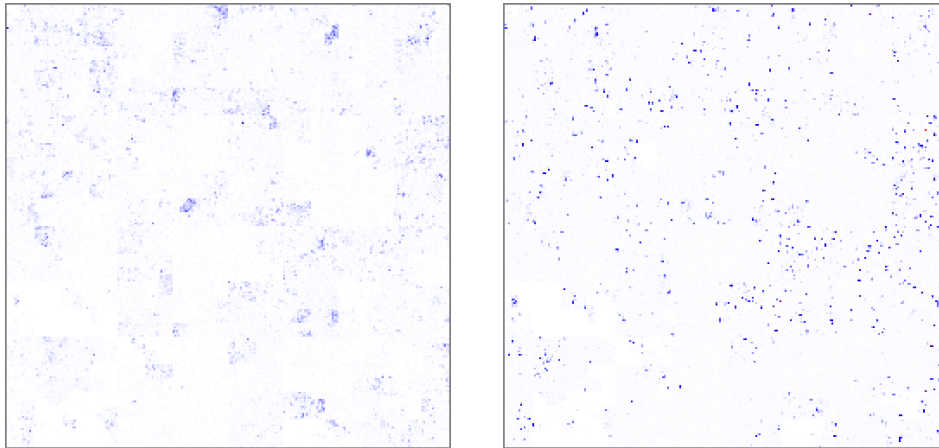
10

Figure 5: Hilbert curve plot of pile-ups for H3K4me1 (left) and H3K4me3 (right) on chromosome 10.

```
> plotLongVector( me3.p10[100000000:101000000], main="Chr 10, H3K3me3", offset=100000000 )
```

[Output: Fig. 4.]

The standard approach would be to export the pile-up vectors into a genome track format such as BED[5] and then use a genome browser such as those on the UCSC or Ensembl web sites, or IGB, to zoom in at many places to get a feeling for the data.

The Hilbert curve plot is an approach to display an as detailed picture of the whole chromosome as possible by letting each pixel of a large square represent a quite short part of the chromosome, coding with its colour for the maximum count in this short stretch, where the pixels are arranged such that neighbouring parts of the chromosome appear next to each other in the square. Furthermore, parts which are not directly neighbouring but are ion close distance should not be separated much in the square either. Fig. 5 shows the two pile-up vectors in this so-called Hilbert curve plot.

In order to understand this plot you need to know how the pixels are arranged to fulfil the requirements just outlined as well as possible. To my knowledge, the first to study this problem in detail and to come up with the solution also used here was D. A. Keim in Ref. [Kei96] (where he used the data to visualise long time-series data of stock-market prices). He went back to an old idea of Peano [Pea90] and Hilbert [Hil91], space-filling curves. Peano astonished the mathematics community at the end of the 19th century by presenting a continuous mapping of a line to a square, i.e., showed that a line can be folded up such that it passes through every point within a square, thus blurring the seemingly clear-cut distinction between one- and two-dimensional objects. Such a space-filling curve is a fractal, i.e., it has infinitely many corners and repeats its overall form in all levels of its details. Fig. 6 shows the first six level of the construction of Hilbert's variant of Peano's curve. Observe how at level k a line of length 2^{2k} passes through each "pixel" of a square of dimension $2^k \times 2^k$, and how this curve is produced connecting four copies (in different orientations) of the curve at the previous level, $k-1$.

Figure 6 has been produced with the function `plotHilbertCurve` which is provided just for demonstration purposes.

```
> library( grid )
```

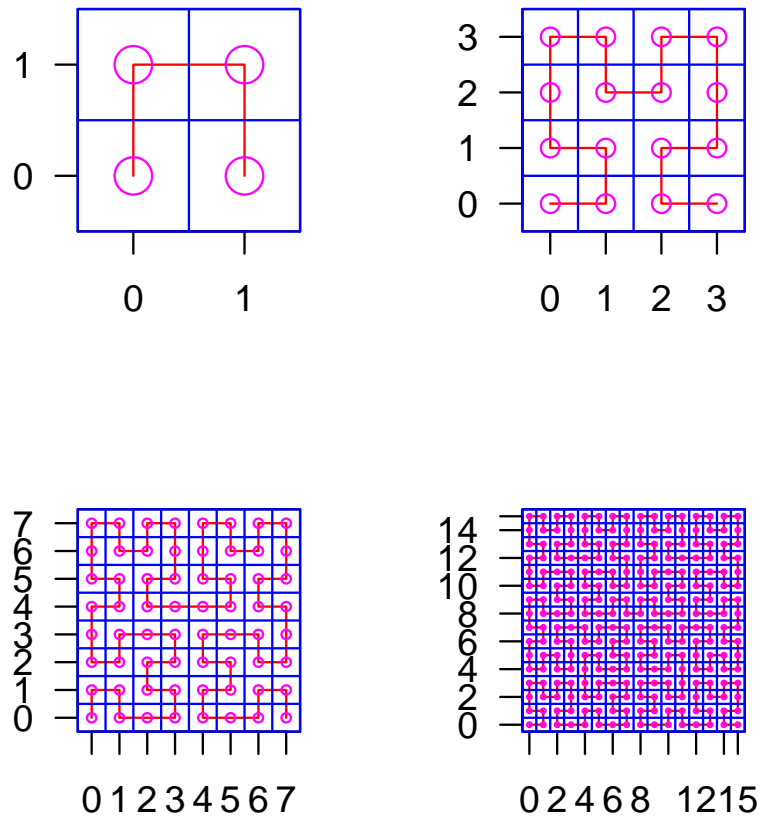[5]A function to do that might be added soon to `ShortRead`.

Figure 6: The first four levels of the Hilbert curve fractal.

```
> pushViewport( viewport( layout=grid.layout( 2, 2 ) ) )
> for( i in 1:4 ) {
+     pushViewport( viewport(
+         layout.pos.row=1+(i-1)%/%2, layout.pos.col=1+(i-1)%%2 ) )
+     plotHilbertCurve( i, new.page=FALSE )
+     popViewport( )
+ }
```

Going back to Fig. 5, we can now see clear difference between the two samples. The following observations my be made just from comparing these two plots: The peaks of H3K4me3 are tall, narrow, and well defined, while those for H3K4me1 are rather washed out. In both cases the peaks spread out over the whole chromosome, but some areas have nearly no signal. These empty parts are the same in both cases. These points were not clear only from Fig. . Exploring the plot interactively as described in the following allows to get considerable more insights.

6.2   The `HilbertVis` GUI

In order to study the pile-up vectors, you can now simply call

```
hilbertDisplay( me1.p10, me3.p10 )
```

A GUI, as depicted in Fig. 7 will pop up that allows you to interactively explore your data in the Hilbert curve plot representation. First, move the mouse over the coloured square and observe how the small red line in the right-hand gauge (labelled "Displayed part of sequence") indicates where within the chromosome you are pointing. Playing with this feature allows you to quickly orient yourself on how the chromosome is folded into the square. You can also read off the exact position from the field "Bin under mouse cursor"[6]

Use the left mouse button to zoom in by clicking on one of the four quarters of the image. You can only zoom into a quarter, not into any part of the image, because this ensures that the displayed part is always a single consecutive stretch of the chromosome. The left-hand gauge (labelled "Full sequence") indicates which part is displayed: the full width of the gauge represents the whole chromosome, the portion highlighted in red the part that is currently displayed in the square. The coordinates of the first and last displayed base are printed in the edges of the right-hand gauge. With the radio buttons labelled "Effect of left mouse button" you my switch from zooming into a quarter to zooming into a 1/64 part, i.e. into one of the small squares in a though 8×8 grid. Use the buttons at the bottom to zoom out.

If you have passed several vectors when calling `hilbertDisplay`, you may switch back and forth between them with the buttons "Next" and "Previous" in order to compare the displayed parts.

The two buttons "Coarser" and "Finer" allow to adjust the pixel size. Initially, each bin is represented by one pixel at your monitor's resolution, and there are 512×512 pixels in the image. Pressing "Coarser" once blows up each image pixel to a 2×2 square of monitor pixels, which allows for easier viewing but reduces the number of displayed image pixels to 256×256, i.e., each pixel now represents four times as many base pairs.

There are a number of optional parameters to `hilbertDisplay` that come in handy, e.g., if you have vectors of differing length, if you want to customise colours or change a few other points. Refer to the help page (displayed with `?hilbertDisplay`) for details.

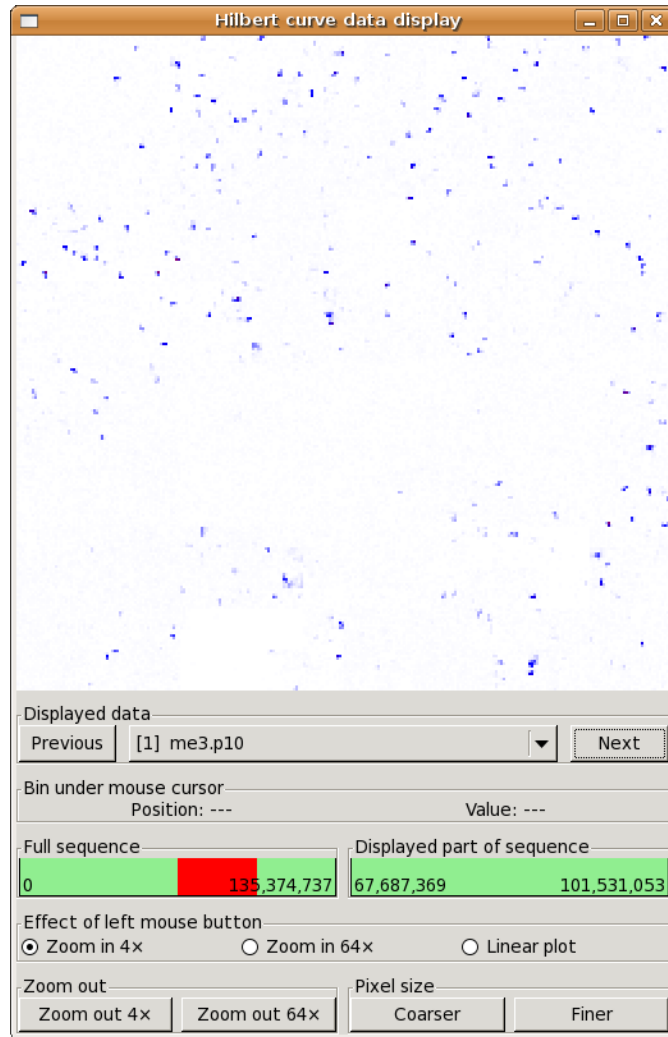
[6]The display of the bin's value is not yet functional.

13

Figure 7: The graphical user interface (GUI) provided by `HilbertVisGUI`.

6.3 The callback interface

If you select the mode "Linear plot" as "Effect of left mouse button" and click somewhere in the plot, a windows pops up with a linear plot that displays the part of the chromosome represented by 256 pixels around the pixel on which you have clicked. (256 pixels correspond quite roughly to the size of the cross-hair mouse cursor). This is useful to get a detailed view of the shape of peaks.

To do the linear plot, HilbertDisplay calls the R function `simpleLinPlot` defined in the `HilbertVisGUI` package. This is a simple wrapper around the function `plotLongVector` discussed earlier. Here is the definition of simpleLinPlot:

```
> simpleLinPlot

function (data, info)
{
    hw <- (info$dispHi - info$dispLo)/1024
    left <- max(1, info$bin - hw)
    right <- min(info$bin + hw, length(data))
    plotLongVector(data[left:right], offset = left, main = info$seqName,
        shrinkLength = min(2 * hw + 1, 4000))
}
<environment: namespace:HilbertVisGUI>
```

You can replace this function by supplying your own plotting function as the argument `plotFun` to `hilbertDisplay`. Your function must take two arguments that should be called `data` and `info`, as above, and will be filled in by `hilbertDisplay` with the displayed vector and information about where the user clicked and which part of the vector is being displayed. Try the following example to see the format of this data:

```
dumpDataInsteadOfPlotting <- function( data, info ) {
    str( data )
    print( info )
}
hilbertDisplay( me1.p10, me3.p10, plotFunc=dumpDataInsteadOfPlotting )
```

Zoom in a bit, then switch to "linear plot" and click somewhere. `dumpDataInsteadOfPlotting` will be called and output such as the following appears on your R console:

```
num [1:135374737] 0 0 0 0 0 0 0 0 0 0 ...
$binLo
[1] 22950198

$bin
[1] 22950262

$binHi
[1] 22950327

$dispLo
[1] 16921843

$dispHi
```

```
[1] 25382764


$seqIdx
[1] 1


$seqName
[1] "me1.p10"
```

See `?hilbertDisplay` for an explanation of those fields that are not self-explanatory.

This feature is meant to allow for customised linear plots (maybe using the `GenomeGraph` package to add annotation) but can also be used for other things than plotting, e.g., calculating some statistics about a peak clicked on.

6.4 Three-channel display

In order to look for spatial correlations in different data vectors, it may be useful to overlay the corresponding Hilbert curve plots in different colours. The function `hilbertDisplayThreeChannel` allows to display three data vectors simultaneously, using the red, green, and blue channel of the displayed image for the first, second, and third, vector. We may want to see whether the areas with strong H3K4me1 occurance are at the same chromosome regions as the majority of the H3K4me3 peaks. Furthermore, we may use the third channel to indicate the presence of exons.

We first obtain a list of all exons on chromosome 10 from EnsEMBL via BioMart:

```
> library( biomaRt )
> ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")
> exons <- getBM( attributes=c( "exon_chrom_start", "exon_chrom_end" ),
+     filters="chromosome_name", values="10", mart=ensembl )
```

Them, we construct a vector that indicates for each base pair on chromosome 10, whether it is exonic or not (bty means of the values 1 and 0).

```
> ex.p10 <- rep( 0, seqlens["10"] )
> for( i in 1:nrow(exons) )
+     ex.p10[ exons$exon_chrom_start[i] : exons$exon_chrom_end[i] ] <- 1
```

With the following command, we get a 3-color representation of the three vectors in the HilbertDisplay GUI:

```
> hilbertDisplayThreeChannel( me1.p10 / 100, me3.p10 / 200, ex.p10 / 3 )
```

See Fig. refthreeColor for the image that the GUI shows. While the function `hilbertDisplay` adjusts to the value range of the data (or can be manually adjusted with optional the `paletteSteps` argument), the function `hilbertDisplayThreeChannel` expects all three vectors to be in the value range between 0 and 1. This range is transformed to colours from black to a saturated red, green, and blue. Values below 0 or above 1 are cut and displayed as if they were 0 or 1. To get the pile-up vectors down to this range, an obvious step would be to devide by their maximum value. However, this gives a too dark value, and hence, I have chosen for Fig. 8 larger scaling factors, allowing extremely high peaks to become saturated.
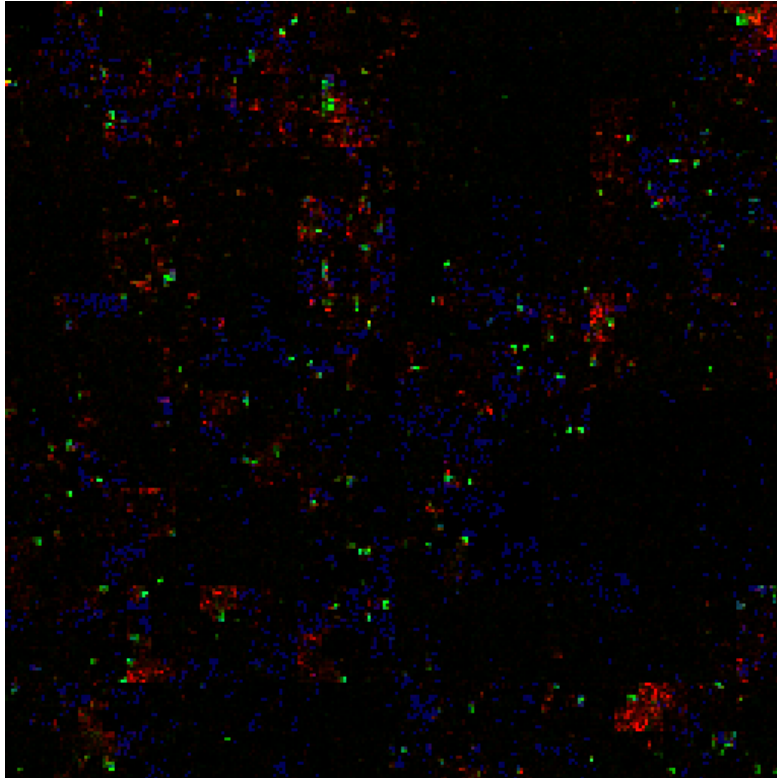
Figure 8: A three-color overlay (obtained with the function `hilbertDisplayThreeChannel`) of Hilbert curves for H3K4me1 (red), H3K4me3 (green) and an exon indication (blue). The image shows a zoom into the first quarter of chromosome 10 (i.e., the top left quarter of the images in Fig. 5).

17

7 Correlation with transcription start

A common plot to do with histone modification ChIP-Seq data is to see how the pile-up correlates with transcription start sites (TSS). This is done quite easily.

First, we get a list of known TSSs on chromosome 10 from EnsEMBL (again via BioMart).

```
> tss <- getBM( attributes=c( "transcript_start", "strand" ),
+    filters="chromosome_name", values="10", mart=ensembl )
```

Then, we go through all TSS, cutting out a window from 200 bp upstreams to 200 bp downstreams of the TSS and sum these up these vectors of length 401.

```
> wnd <- -200:200
> tme1 <- rep( 0, length(wnd) )
> tme3 <- rep( 0, length(wnd) )
> for( i in 1:nrow(tss) ) {
+    tme1 <- tme1 + me1.p10[ tss$transcript_start[i] + wnd * tss$strand[i] ]
+    tme3 <- tme3 + me3.p10[ tss$transcript_start[i] + wnd * tss$strand[i] ] }
```

Note how the strand information (which is an integer, either 1 or −1) is used to reverse the window for the '−' strand.

Normally, one would add all the other chromosomes, as well. For this vignette, we simply plot what we have so far:

```
> matplot( wnd, cbind( tme1, tme3 ),
+    type="l", col=c("red","green"), lty="solid", xlab="", ylab="" )
> abline( v=0, col="gray" )
```

[Output: Fig. 9.]

Session info

```
> sessionInfo()

R version 2.8.0 Under development (unstable) (2008-09-13 r46541)
x86_64-unknown-linux-gnu

locale:
LC_CTYPE=en_GB.UTF-8;LC_NUMERIC=C;LC_TIME=en_GB.UTF-8;LC_COLLATE=en_GB.UTF-8;LC_MONETARY=C;LC_MES

attached base packages:
[1] grid       tools     stats     graphics  grDevices utils     datasets
[8] methods    base

other attached packages:
[1] biomaRt_1.15.2      HilbertVisGUI_0.99.5 HilbertVis_0.99.5
[4] ShortRead_0.1.56    lattice_0.17-15      Biostrings_2.9.79
[7] Biobase_2.1.7       IRanges_0.99.16

loaded via a namespace (and not attached):
[1] RCurl_0.9-4 XML_1.96-0
```
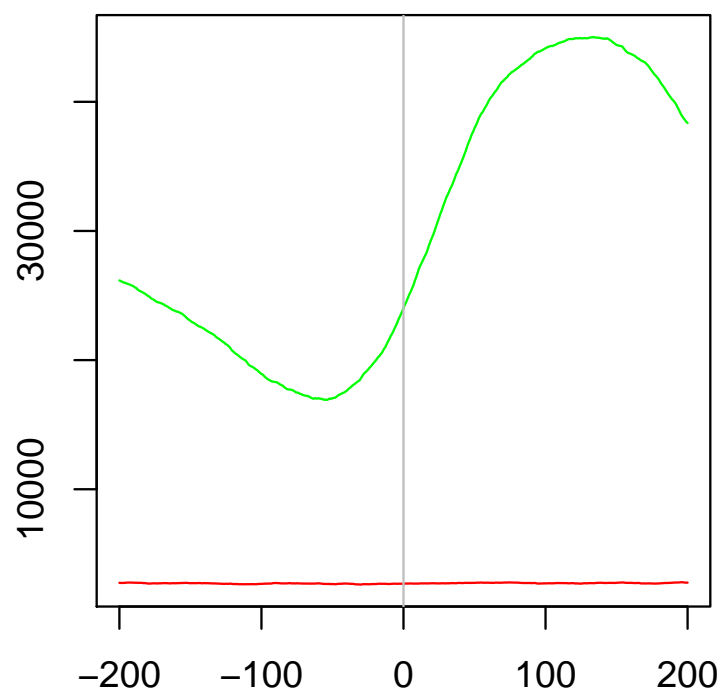
Figure 9: Correlation against transcription start sites for H3K4me1 (red) and H3K4me3 (green).

References

[BCC+07] A. Barski, S. Cuddapah, K. Cui, T.-Y. Roh, D. E. Schones, Z. Wang, G. Wei, I. Chepelev, K. Zhao. *High-resolution profiling of histone methylations in the human genome.* Cell **129** (2007), 823.

[Hil91] D. Hilbert. *Über stetige Abbildungen einer Linie auf ein Flächenstück.* Mathematische Annalen **38** (1891), 459.

[Kei96] D. A. Keim. *Pixel-oriented visualization techniques for exploring very large data bases.* J. Comp. Graph. Stat. **5** (1996), 58.

[Li] H. Li. *Maq: Mapping with alignment qualities.* [web link].

[Mor] M. Morgan. *ShortRead: Base classes and methods for high-throughput short-read sequencing data.* R package version 0.1.23.

[Pea90] G. Peano. *Sur une courbe qui rempli toute une aire plaine.* Mathematische Annalen **36** (1890), 157.