segmentSeq: methods for identifying small RNA loci from high-throughput sequencing data

Thomas J. Hardcastle

April 10, 2013

1 Introduction

High-throughput sequencing technologies allow the production of large volumes of short sequences, which can be aligned to the genome to create a set of *matches* to the genome. By looking for regions of the genome which to which there are high densities of matches, we can infer a segmentation of the genome into regions of biological significance. The methods we propose allows the simultaneous segmentation of data from multiple samples, taking into account replicate data, in order to create a consensus segmentation. This has obvious applications in a number of classes of sequencing experiments, particularly in the discovery of small RNA loci and novel mRNA transcriptome discovery.

We approach the problem by considering a large set of potential *segments* upon the genome and counting the number of tags that match to that segment in multiple sequencing experiments (that may or may not contain replication). We then adapt the empirical Bayesian methods implemented in the `baySeq` package [1] to establish, for a given segment, the likelihood that the count data in that segment is similar to background levels, or that it is similar to the regions to the left or right of that segment. We then rank all the potential segments in order of increasing likelihood of similarity and reject those segments for which there is a high likelihood of similarity with the background or the regions to the left or right of the segment. This gives us a large list of overlapping segments. We reduce this list to identify non-overlapping loci by choosing, for a set of overlapping segments, the segment which has the lowest likelihood of similarity with either background or the regions to the left or right of that segment and rejecting all other segments that overlap with this segment. For fuller details of the method, see Hardcastle *et al.* [2].

2 Preparation

We begin by loading the `segmentSeq` package.

```
> library(segmentSeq)
```

Note that because the experiments that `segmentSeq` is designed to analyse are usually massive, we should use (if possible) parallel processing as implemented by the `snow` package. We therefore need to load the `snow` package (if it exists) and define a *cluster*.

```
>   library(snow)
> cl <- makeCluster(8, "MPI")
```

If `snow` is not present, we can proceed anyway with a `NULL` cluster. Results may be slightly different depending on whether or not a cluster is used owing to the non-deterministic elements of the method.

```
>   cl <- NULL
```

There is a convenience function, `readGeneric` which is able to read in tab-delimited files which have appropriate column names, and create an `alignmentData` object. Alternatively, if the appropriate column names are not present, we can specify which columns to use for the data. In either case, we pass a character vector of files, together with information on which data are to be treated as replicates to the function. We also need to define the lengths of the chromosome and specifiy the chromosome names as a character. The data here, drawn from text files in the 'data' directory of the `segmentSeq` package are taken from the first million bases of an alignment to chromosome 1 and the first five hundred thousand bases of an alignment to chromosome 2 of *Arabidopsis thaliana* in a sequencing experiment where libraries 'SL9' and 'SL10' are replicates, as are 'SL26' and 'SL32'. Libraries 'SL9' and 'SL10' are sequenced from an Argonaute 6 IP, while 'SL26' and 'SL32' are an Argonaute 4 IP.

```
> chrlens <- c(1e6, 2e5)
> datadir <- system.file("extdata", package = "segmentSeq")
> libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")
> libnames <- c("SL9", "SL10", "SL26", "SL32")
> replicates <- c("AGO6", "AGO6", "AGO4", "AGO4")
> aD <- readGeneric(files = libfiles, dir = datadir,
+                   replicates = replicates, libnames = libnames,
+                   chrs = c(">Chr1", ">Chr2"), chrlens = chrlens,
+                   polyLength = 10, header = TRUE, gap = 200)
> aD

An object of class "alignmentData"
13765 rows and 4 columns

Slot "alignments":
GRanges with 13765 ranges and 4 metadata columns:
          seqnames           ranges strand   |                     tag
             <Rle>        <IRanges>  <Rle>   |                   <Rle>
      [1]    >Chr1       [265, 284]      -   |     AAATGAAGATAAACCATCCA
      [2]    >Chr1       [405, 427]      -   |  AAGGAGTAAGAATGACAATAAAT
      [3]    >Chr1       [406, 420]      -   |           AAGAATGACAATAAA
      [4]    >Chr1       [600, 623]      +   | AAGGATTGGTGGTTTGAAGACACA
      [5]    >Chr1       [665, 688]      +   | ATCCTTGTAGCACACATTTTGGCA
      ...      ...              ...    ... ...                      ...
  [13761]    >Chr2 [179972, 179993]      +   |   ATGAATGGCTCTCTCTAGCGGA
  [13762]    >Chr2 [179978, 180000]      -   |  GAGATTCTCCGCTAGAGAGAGCC
  [13763]    >Chr2 [179999, 180022]      -   | ATTAATATTAATTCATCGGGAAGA
  [13764]    >Chr2 [180002, 180022]      -   |    ATTAATATTAATTCATCGGGA
```

```
  [13765]    >Chr2 [180014, 180037]      +   | AATATTAATGGTATTTGTGGAAAA
           matches chunk  chunkDup
         <numeric> <Rle> <logical>
      [1]         1     1     FALSE
      [2]         1     1     FALSE
      [3]         1     1     FALSE
      [4]         1     1     FALSE
      [5]         1     1     FALSE
      ...       ...   ...       ...
  [13761]         1   279     FALSE
  [13762]         1   279     FALSE
  [13763]         1   279     FALSE
  [13764]         1   279     FALSE
  [13765]         1   279     FALSE
  ---
  seqlengths:
     >Chr1   >Chr2
   1000000  200000

Slot "data":
DataFrame with 5 rows and 4 columns
    SL9  SL10  SL26  SL32
  <Rle> <Rle> <Rle> <Rle>
1     1     0     0     0
2     0     0     0     2
3     0     1     0     0
4     0     1     0     0
5     7     1     0     0
13760 more rows...

Slot "libnames":
[1] "SL9"  "SL10" "SL26" "SL32"

Slot "libsizes":
[1] 4447 6531 9666 6675

Slot "replicates":
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6
```

Next, we process this `alignmentData` object to produce a `segData` object. This `segData` object contains a set of potential segments on the genome defined by the start and end points of regions of overlapping alignments in the `alignmentData` object. It then evaluates the number of tags that hit in each of these segments.

```
> sD <- processAD(aD, gap = 100, cl = cl)
> sD

An object of class "segData"
14444 rows and 4 columns
```

```
Slot "data":
DataFrame with 5 rows and 4 columns
    SL9  SL10  SL26  SL32
  <Rle> <Rle> <Rle> <Rle>
1     1     0     0     0
2     0     1     0     2
3     0     1     0     0
4     7     2     0     0
5    30    28    51    83
14439 more rows...

Slot "libsizes":
[1] 4447 6531 9666 6675

Slot "replicates":
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6

Slot "coordinates":
GRanges with 14444 ranges and 0 metadata columns:
          seqnames           ranges strand
             <Rle>        <IRanges>  <Rle>
      [1]    >Chr1       [265, 284]      *
      [2]    >Chr1       [405, 427]      *
      [3]    >Chr1       [600, 623]      *
      [4]    >Chr1       [600, 688]      *
      [5]    >Chr1       [600, 830]      *
      ...      ...              ...    ...
  [14440]    >Chr2 [179708, 179872]      *
  [14441]    >Chr2 [179708, 180037]      *
  [14442]    >Chr2 [179738, 179872]      *
  [14443]    >Chr2 [179738, 180037]      *
  [14444]    >Chr2 [179923, 180037]      *
  ---
  seqlengths:
     >Chr1   >Chr2
   1000000  200000
```

We can now construct a segment map from these potential segments.

Segmentation by Clustering

A fast method of segmentation can be achieved by exploiting the bimodality of the densities of small RNAs in the potential segments. In this approach, we assign each potential segment to one of two clusters for each replicate group, either as a segment or a null based on the density of sequence tags within that segment. We then combine these clusterings for each replicate group to gain a consensus segmentation map.

```
> clustSegs <- heuristicSeg(sD = sD, aD = aD, RKPM = 300, largeness = 1e8, cl = cl)
```

```
..
```

```
> clustSegs
```

```
GRanges with 774 ranges and 0 metadata columns:
        seqnames           ranges strand
           <Rle>        <IRanges>  <Rle>
    [1]    >Chr1       [  1, 264]      *
    [2]    >Chr1       [265, 284]      *
    [3]    >Chr1       [285, 404]      *
    [4]    >Chr1       [405, 427]      *
    [5]    >Chr1       [428, 599]      *
    ...      ...              ...    ...
  [770]    >Chr2 [178637, 179096]      *
  [771]    >Chr2 [179097, 179111]      *
  [772]    >Chr2 [179112, 179707]      *
  [773]    >Chr2 [179708, 180037]      *
  [774]    >Chr2 [180038, 200000]      *
  ---
  seqlengths:
     >Chr1   >Chr2
   1000000  200000
An object of class "lociData"
774 rows and 4 columns

Slot "replicates"
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6

Slot "libsizes"
AGO6.1 AGO6.2 AGO4.1 AGO4.2
  4447   6531   9666   6675

Slot "groups":
[[1]]
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6


Slot "data":
     AGO6.1 AGO6.2 AGO4.1 AGO4.2
[1,]      0      0      0      0
[2,]      1      0      0      0
[3,]      0      0      0      0
[4,]      0      1      0      2
[5,]      0      0      0      0
769 more rows...

Slot "annotation":
data frame with 0 columns and 774 rows
```

```
Slot "locLikelihoods" (stored on log scale):
           AGO4       AGO6
[1,] 0.01451211 0.02129373
[2,] 0.07159889 0.81662143
[3,] 0.02437336 0.03891005
[4,] 0.41615355 0.79927895
[5,] 0.01914445 0.02856721
769 more rows...
```

Segmentation by Classification

A more refined approach to the problem uses an existing segment map (or, if not provided, a segment map defined by the `clustSegs` function) to acquire empirical distributions on the density of sequence tags within a segment. We can then estimate posterior likelihoods for each potential segment as being either a true segment or a null. We then identifying all potential segments in the with a posterior likelihood of being a segment greater than some value 'locsens' and containing no subregion with a posterior likelihood of being a null greater than 'nulsens'. We then greedily select the longest segments satisfying these criteria that do not overlap with any other such segments in defining our segmentation map.

```
> classSegs <- classifySeg(sD = sD, aD = aD, cD = clustSegs,
+                          subRegion = NULL, getLikes = TRUE,
+                          lociCutoff = 0.9, nullCutoff = 0.9, cl = cl)

........

> classSegs

GRanges with 300 ranges and 0 metadata columns:
        seqnames           ranges strand
           <Rle>        <IRanges>  <Rle>
    [1]    >Chr1   [    1,   599]      *
    [2]    >Chr1   [  600,   967]      *
    [3]    >Chr1   [  968, 17054]      *
    [4]    >Chr1   [17055, 18728]      *
    [5]    >Chr1   [18729, 27656]      *
    ...      ...              ...    ...
  [296]    >Chr2 [169231, 178343]      *
  [297]    >Chr2 [178344, 178636]      *
  [298]    >Chr2 [178637, 179707]      *
  [299]    >Chr2 [179708, 180037]      *
  [300]    >Chr2 [180038, 200000]      *
  ---
  seqlengths:
     >Chr1   >Chr2
   1000000  200000
An object of class "lociData"
300 rows and 4 columns
```

```
Slot "replicates"
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6

Slot "libsizes"
AGO6.1 AGO6.2 AGO4.1 AGO4.2
  4447   6531   9666   6675

Slot "groups":
[[1]]
[1] AGO6 AGO6 AGO4 AGO4
Levels: AGO4 AGO6


Slot "data":
     AGO6.1 AGO6.2 AGO4.1 AGO4.2
[1,]      1      1      0      2
[2,]     54     46     65     83
[3,]      2      3      0      0
[4,]    682    621   1405   1103
[5,]      0      3      0      0
295 more rows...

Slot "annotation":
data frame with 0 columns and 300 rows

Slot "locLikelihoods" (stored on log scale):
            AGO4        AGO6
[1,] 0.074013105 0.010982005
[2,] 0.980236146 0.985699836
[3,] 0.008054725 0.001786787
[4,] 0.999878075 0.999125103
[5,] 0.007645485 0.010171339
295 more rows...
```

By one of these methods, we finally acquire an annotated `countData` object, with the annotations describing the co-ordinates of each segment.

We can use this `countData` object, in combination with the `alignmentData` object, to plot the segmented genome.

```
> par(mfrow = c(2,1), mar = c(2,6,2,2))
> plotGenome(aD, clustSegs, chr = ">Chr1", limits = c(1, 1e5),
+            showNumber = FALSE, cap = 50)
> plotGenome(aD, classSegs, chr = ">Chr1", limits = c(1, 1e5),
+            showNumber = FALSE, cap = 50)
```

This `countData` object can now be examined for differential expression with the `baySeq` package.

References

[1] Thomas J. Hardcastle and Krystyna A. Kelly. *baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data.* BMC Bioinformatics (2010)

[2] Thomas J. Hardcastle and Krystyna A. Kelly and David C. Baulcombe. *Identifying small RNA loci from high-throughput sequencing data.* In press (2011)

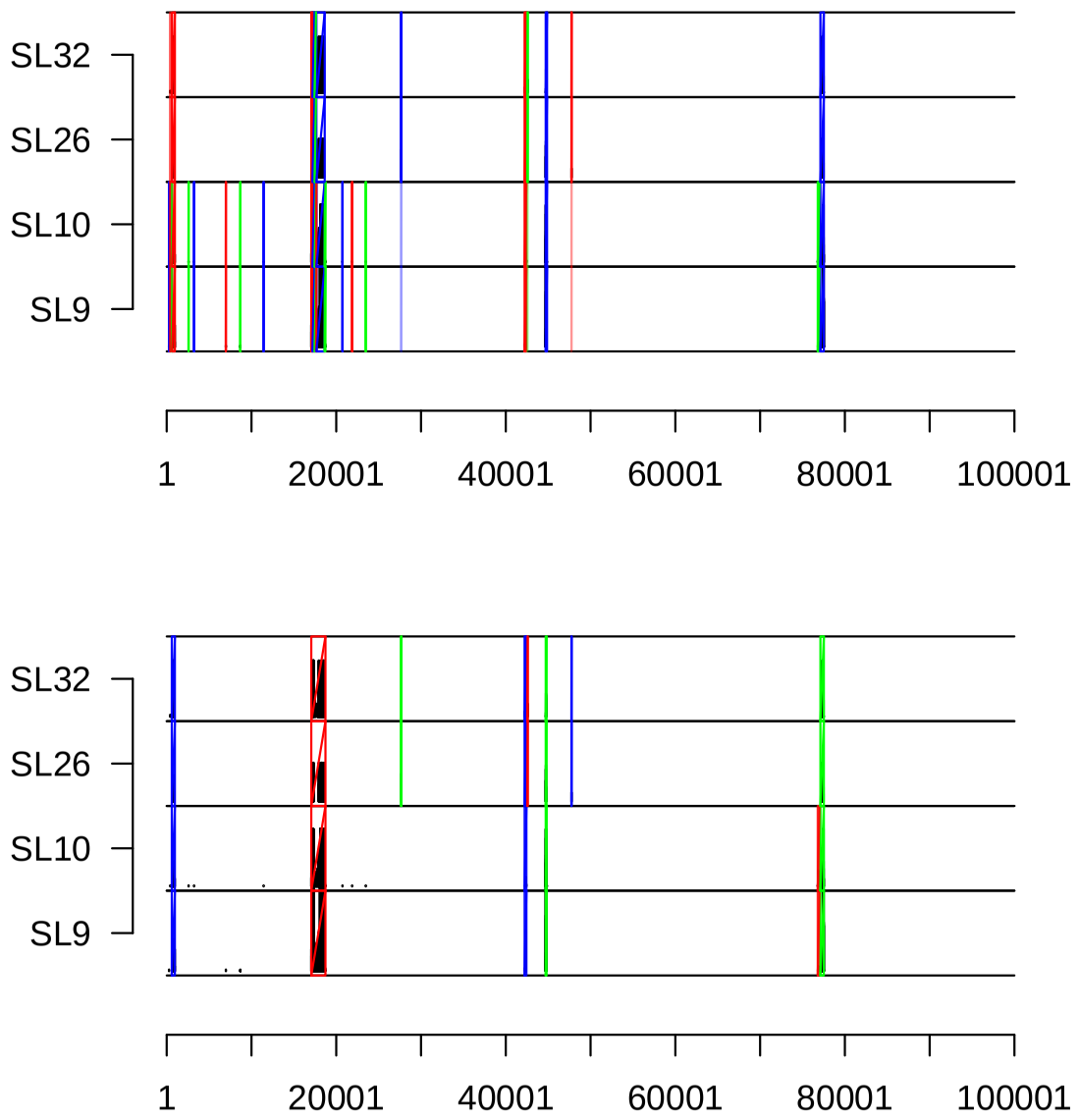

Figure 1: The segmented genome (first 10^5 bases of chromosome 1.