Biostrings Lab (ChIP-seq course, Nov. 2008)

November 14, 2008

1 Lab overview

Learn the basics of Biostrings and the *BSgenome data packages*.

For this lab you need:

- A laptop with the latest release version of R (R 2.8 series).
- The Biostrings, BSgenome and BSgenome.Mmusculus.UCSC.mm9 packages.
- `topReads.rda`: serialized object containing the top 1000 reads for each lane of 2 ChIP-seq experiments.

2 Check your installation

Exercise 1

1. *Start R and load the BSgenome.Mmusculus.UCSC.mm9 package.*
2. *Display chromosome 1.*
3. *Load* `topReads.rda`*.*
4. *Display a few reads from experiment 2 / lane 1.*

3 Basic containers

3.1 DNAString objects

The *DNAString* class is the basic container for storing a large nucleotide sequence. Unlike a standard character vector in R that can store an arbitrary number of strings, a *DNAString* object can only contain 1 sequence. Like for most classes defined in Biostrings, `DNAString` is also the name of the constructor function for *DNAString* objects.

Exercise 2

1. *Pick up a read from experiment 2 / lane 1 and convert it to a DNAString object (use the* `DNAString` *constructor).*

2. *Use* `length` *and* `alphabetFrequency` *on it.*

3. *Get its reverse complement.*

4. *Extract an arbitrary substring with* `subseq`*.*

3.2 DNAStringSet objects

The *DNAStringSet* class is the basic container for storing an arbitrary number of nucleotide sequences. Like with R character vectors (and any vector-like object in general), use `length` to get the number of elements (sequences) stored in a *DNAStringSet* object and the `[` operator to subset it. In addition, subsetting operator `[[` can be used to extract an arbitrary element as a *DNAString* object.

Exercise 3

1. *Use the* `DNAStringSet` *constructor to store the 1000 reads from experiment 2 / lane 1 into a DNAStringSet object. Let's call this instance* `dict0`*.*

2. *Use* `length` *and* `width` *on* `dict0`*.*

3. *Use subsetting operator* `[` *to remove its 2nd element.*

4. *Invert the order of its elements.*

5. *Use subsetting operator* `[[` *to extract its 1st element as a DNAString object.*

6. *Use the* `DNAStringSet` *constructor (i) to remove the last 2 nucleotides of each element, then (ii) to keep only the last 10 nucleotides.*

7. *Call* `alphabetFrequency` *on* `dict0` *and on its reverse complement. Try again with* `collapse=TRUE`*.*

8. *Remove reads with Ns (put the "clean" dictionary in* `dict0` *again).*

3.3 XStringViews objects

An *XStringViews* object contains a set of views on the same sequence called *the subject* (for example this can be a *DNAString* object). Each view is defined by its start and end locations: both are integers such that start <= end. The `Views` function can be used to create an *XStringViews* object given a subject and a set of start and end locations. Like for *DNAStringSet* objects, `length`, `width`, `[` and `[[` are supported for *XStringViews* objects. Additional `subject`, `start`, `end` and `gaps` methods are also provided.

Exercise 4

1. *Use the* `Views` *function to create an XStringViews object with a DNAString subject. Make it such that some views are overlapping but also that the set of views don't cover the subject entirely.*

2. *Try* `subject`, `start`, `end` *and* `gaps` *on this XStringViews object.*

3. *Try* `alphabetFrequency` *on it.*

4. *Turn it into a DNAStringSet object with the* `DNAStringSet` *constructor.*

4 BSgenome data packages

The name of a *BSgenome data package* is made of 4 parts separated by a dot (e.g. BSgenome.Celegans.UCSC.ce2):

- The 1st part is always `BSgenome`.
- The 2nd part is the name of the organism (abbreviated).
- The 3rd part is the name of the organisation who assembled the genome.
- The 4th part is the release string or number used by this organisation for this assembly of the genome.

All *BSgenome data package* contain a single top level object whose name matches the second part of the package name.

Exercise 5

1. *Load BSgenome.Mmusculus.UCSC.mm9 and display its top level object. Note that this doesn't load any sequence in memory yet.*

2. *Use* `seqlengths` *on it to get the lengths of the single sequences (this doesn't load any sequence either).*

3. *Display some of the chromosomes. Some information about the built-in masks is displayed. Let's drop the masks for now by accessing the sequences with e.g.* `unmasked(Mmusculus$chrM)`*. Note that a sequence is not loaded until it is accessed.*

4. *Do the chromosomes contain IUPAC extended letters?*

5. *Use* `chartr` *to simulate a bisulfite transformation of chromosome 1 (see* `?chartr`*).*

5 String matching

5.1 The matchPattern function

This function finds all the occurences (aka *matches* or *hits*) of a given pattern in a reference sequence called *the subject*.

Exercise 6

1. *Find all the matches of a short pattern (invent one) in mouse chromosome 1. Don't choose the pattern too short or too long.*

2. *In fact, if we don't take any special action, we only get the hits in the plus strand of the chromosome. Find the matches in the minus strand too. (Note: the cost of taking the reverse complement of an entire chromosome sequence can be high in terms of memory usage. Try to do something better.)*

3. `matchPattern` *now support indels (recent improvement) via the* `with.indels` *argument. Use the same pattern to find all the matches in chromosome 1 that are at an edit distance <= 2 from it.*

5.2 The vmatchPattern function

This function finds all the matches of a given pattern in a set of reference sequences.

Exercise 7

1. *Load the* `upstream5000` *object from* `Mmusculus` *and find all the matches of a short arbitrary pattern in it.*

2. *The value returned by* `vmatchPattern` *is an MIndex object containing the match coordinates for each reference sequence. You can use the* `startIndex` *and* `endIndex` *accessors on it to extract the match starting and ending positions as lists (one list element per reference sequence).* `[[` *extracts the matches of a given reference sequence as an MIndex object.* `coundIndex` *extract the match counts as an integer vector (one element per reference sequence).*

5.3 Ambiguities

IUPAC extended letters can be used to express ambiguities in the pattern or in the subject of a search with `matchPattern`. This is controlled via the `fixed` argument of the function. If `fixed` is `TRUE` (the default), all letters in the pattern and the subject are interpreted litterally. If `fixed` is `FALSE`, IUPAC extended letters in the pattern and in the subject are interpreted as ambiguities e.g. `M` will match `A` or `C` and `N` will match any letter (the `IUPAC_CODE_MAP` named character vector gives the mapping between IUPAC letters and the set of nucleotides that they stand for). The most common use of this feature is to introduce wildcards in the pattern by replacing some of its letters with `N`s.

Exercise 8

1. *Search pattern* `GAACTTTGCCACTC` *in Mouse chromosome 1.*

2. *Repeat but this time allow the 2nd* `T` *in the pattern (6th letter) to match anything. Anything wrong?*

3. *Call* `matchPattern` *with* `fixed="subject"` *to work around this problem.*

5.4 Masking

The *MaskedDNAString* container is dedicated to the storage of masked DNA sequences. As mentioned previously, you can use the `unmasked` accessor to turn a *MaskedDNAString* object into a *DNAString* object (the masks will be lost), or use the `masks` accessor to extract the masks (the sequence that is masked will be lost).

Each mask on a sequence can be active or not. Masks can be activated individually with:

```
> chr1 <- Mmusculus$chr1
> active(masks(chr1))["TRF"] <- TRUE # activate Tandem Repeats Finder mask
```

or all together with:

```
> active(masks(chr1)) <- TRUE # activate all the masks
```

Some functions in Biostrings like `alphabetFrequency` or the string matching functions will skip the masked region when walking along a sequence with active masks.

Exercise 9

1. *What percentage of Mouse chromosome 1 is made of assembly gaps?*
2. *Check the alphabet frequency of Mouse chromosome 1 when only the AGAPS mask is active, when only the AGAPS and AMB masks are active. Compare with unmasked chromosome 1.*
3. *Try* `as(chr1 , "XStringViews")` *and* `gaps(as(chr1 , "XStringViews"))` *with different sets of active masks. How do you use this to display the contigs as views?*
4. *Activate all masks and find the occurences of an arbitrary DNA pattern in it. Compare to what you get with unmasked chromosome 1.*

In addition to the built-in masks, the user can put its own mask on a sequence. Two types of user-controlled masking are supported: *by content* or *by position*. The `maskMotif` function will mask the regions of a sequence that contain a motif specified by the user. The `Mask` constructor will return the mask made of the regions defined by the start and end locations specified by the user (like with the `Views` function).

5.5 Finding the hits of a large set of short motifs

Our own competitor to other fast alignment tools like MAQ or bowtie is the `matchPDict` function. Its speed is comparable to the speed of MAQ but it uses more memory than MAQ to align the same set of reads against the same genome. Here are some important differences between `matchPDict` and MAQ (or bowtie):

1. `matchPDict` ignores the quality scores,
2. it finds all the matches,
3. it fully supports 2 or 3 (or more) mismatching nucleotides anywhere in the reads (performance will decrease significantly though if the reads are not long enough),
4. it supports masking (masked regions are skipped),
5. it supports IUPAC ambiguities in the subject (useful for SNP detection).

The workflow with `matchPDict` is the following:

1. Preprocess the set of short reads with the `PDict` constructor.
2. Call `matchPDict` on it.
3. Query the *MIndex* object returned by `matchPDict`.

Exercise 10

1. *Preprocess* `dict0` *(obtained earlier from* `topReads.rda`*) with the* `PDict` *constructor.*
2. *Use this PDict object to find the (exact) hits of* `dict0` *in Mouse chromosome 1.*
3. *Use* `countIndex` *on the MIndex object returned by* `matchPDict` *to extract the nb of hits per read.*
4. *Which read has the highest number of hits? Display those hits as an XStringViews object. Check this result with a call to* `matchPattern`*.*
5. *You only got the hits that belong to the + strand. How would you get the hits that belong to the - strand?*
6. *Redo this analysis for inexact matches with at most 2 mismatches per read in the last 20 nucleotides.*