Biostrings/BSgenome Lab (BioC2009)

Hervé Pagès and Patrick Aboyoun
Fred Hutchinson Cancer Research Center
Seattle, WA

July 26, 2009

1 Lab overview

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

For this lab you need:

- A laptop with the current devel version of R (R 2.10 series).
- The Biostrings, drosophila2probe, BSgenome, BSgenome.Celegans.UCSC.ce2 and BSgenome.Dmelanogaster.UCSC.dm3 packages.

2 Check your installation

Exercise 1

1. *Start R and load the BSgenome.Dmelanogaster.UCSC.dm3 package.*
2. *Display chromosome 2L.*

3 Basic string 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. *Load the BSgenome.Celegans.UCSC.ce2 package and display chromosome I. Use the* `class` *function on this chromosome to see the type of container used for its storage.*
2. *Use* `length` *and* `alphabetFrequency` *on it.*

3. *Extract an arbitrary subsequence with* `subseq`.

4. *Get the reverse complement of this subsequence.*

3.2 DNAStringSet objects

The *DNAStringSet* class is the basic container for storing an arbitrary number of nucleotide sequences. The `length` of a *DNAStringSet* object is the number of sequences in it. The `[` operator can be used to subset it i.e. to select some of the sequences. The `[[` operator can be used to extract an arbitrary sequence as a *DNAString* object.

Exercise 3

1. *The* drosophila2probe *package contains the probe sequence for the Drosophila 2 microarray from Affymetrix. Load this package and display the first 5 probe sequences stored in the* `drosophila2probe` *object.*

2. *Use the* `DNAStringSet` *constructor to store all the probe sequences into a DNAStringSet object. Let's call this object* `dict0`.

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

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

5. *Invert the order of its elements.*

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

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

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

9. *How many probes have a GC content of 80% or more?*

10. *What's the GC content for the entire microarray?*

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. `length`, `width`, `[` and `[[` are supported for *XStringViews* objects, just like for *DNAStringSet* objects. In addition, `subject`, `start`, `end` and `gaps` methods are also provided for *XStringViews* objects.

Exercise 4

1. *Use the* `Views` *function to create an XStringViews object on Worm chromosome I. Make it such that some views are overlapping but also that the set of views doesn't cover the subject entirely.*

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

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

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

3.4 MaskedDNAString objects

A *MaskedDNAString* object contains a masked DNA sequence, that is, a *DNAString* object plus a set of masks. The purpose of these masks is to allow the user to mask the regions that need to be ignored during some computations.

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).

Exercise 5

1. *Load the* BSgenome.Dmelanogaster.UCSC.dm3 *and display chromosome 2L.*

2. *Get rid of the masks defined on this chromosome.*

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

```
> chr2L <- Dmelanogaster$chr2L
> active(masks(chr2L))["TRF"] <- TRUE # activate Tandem Repeats Finder mask
```

or all together with:

```
> active(masks(chr2L)) <- TRUE # activate all the masks
```

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

Exercise 6

1. *What percentage of Fly chromosome X is made of assembly gaps?*

2. *Confirm this result by checking the alphabet frequency of unmasked chromosome X.*

3. *Try* `as(chrX , "XStringViews")` *and* `gaps(as(chrX , "XStringViews"))` *on masked chromosome X. What are the lengths of the assembly gaps?*

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).

4 BSgenome data packages

You've already used the *BSgenome data packages* for Worm and Fly. The Bioconductor project provides *BSgenome data packages* for the commonly studied organism. Use `available.genomes()` to see all the packages available.

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 7

1. *Get the list of all available BSgenome data packages.*
2. *After you've loaded a BSgenome data package, use* `?<name-of-the-package>` *to see useful information about the package and some examples on how to use it.*
3. *What's the quick and easy way to get the lengths of all the sequences stored in a BSgenome data package?*

In a given *BSgenome data package*, either all DNA sequences are masked or none is. In the former case, the sequences are always masked with 4 built-in masks:

- the masks of assembly gaps, aka "the AGAPS masks";
- the masks of intra-contig ambiguities, aka "the AMB masks";
- the masks of repeat regions that were determined by the RepeatMasker software, aka "the RM masks";

- the masks of repeat regions that were determined by the Tandem Repeats Finder software (where only repeats with period less than or equal to 12 were kept), aka "the TRF masks".

If there is no *BSgenome data package* for your organism, then you can make your own package. This process is described in the BSgenomeForge vignette from the *BSgenome* software package.

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 8

1. *Find all the matches of a short pattern (invent one) in Worm chromosome I. 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. *Use the* `max.mismatch` *argument to find all the matches in chromosome I that have at most 1 mismatching nucleotide.*

4. *Use the* `max.mismatch` *argument together with the* `with.indels` *argument to find all the matches in chromosome I that are at an edit distance <= 2 from the pattern.*

5.2 The vmatchPattern function

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

Exercise 9

1. *Load the* `upstream1000` *object from* `Dmelanogaster` *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.* `countIndex` *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 10

1. *Search pattern* `GAACTTTGCCAC` *in Celegans chromosome I.*
2. *Repeat but this time allow the 3* `T`*s in the pattern to match anything.*

5.4 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 11

1. *Preprocess* `dict0` *(containing the probe sequences from Affymetrix Drosophila 2, see exercise 3) with the* `PDict` *constructor.*
2. *Use this PDict object to find the (exact) hits of* `dict0` *in unmasked Fly chromosome 2L.*

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 using inexact matching: now we want to allow up to 2 mismatching nucleotides per probe in the last 12 nucleotides of the probe.*