Biostrings Lab (BioC2008)

H. Pagès
Gentleman Lab
Fred Hutchinson Cancer Research Center
Seattle, WA

July 31, 2008

1 Lab overview

1.1 Goals

Learn the basics of the Biostrings package and how to use it together with the *BSgenome data packages* for different kinds of genome-wide sequence analyses.

1.2 Prerequisites

For this lab you need:

- A laptop with the latest release version of R (R 2.7 series).
- The following Bioconductor packages (installed with `biocLite()`):
  - Biostrings
  - BSgenome
  - hgu95av2probe
  - drosophila2probe
  - BSgenome.Hsapiens.UCSC.hg18
  - BSgenome.Dmelanogaster.UCSC.dm3
  - BSgenome.Celegans.UCSC.ce2
  - SNPlocs.Hsapiens.dbSNP.20071016
- Some familiarity with R and object oriented programming.

1.3 Useful links and additional resources

- Bioconductor website: http://bioconductor.org/
- This document and other material used for this lab (including solutions to the exercises): http://bioconductor.org/workshops/2008/BioC2008/labs/biostrings/
- The Bioconductor mailing lists: http://bioconductor.org/docs/mailList.html

2 Introduction

2.1 Biostrings

The Biostrings package provides:

- The infrastructure for representing and manipulating large nucleotide sequences (up to hundreds of millions of letters) in Bioconductor. This infrastructure consists of a class system (the *DNAString*, *DNAStringSet*, *XStringViews* containers, etc...), plus a set of constructors for creating objects belonging to these classes, plus a set of accessor methods for querying or modifying the content of these objects.
- Utility functions for extracting basic sequence information (`alphabetFrequency`, `dinucleotideFrequency` functions, etc...), or for performing basic sequence transformations (`reverseComplement`, `chartr`, `injectHardMask` functions, etc...).
- String matching functions for finding all the occurrences (hits) of arbitrary motifs in a reference sequence (`matchPattern`, `matchProbePair`, `matchPWM`, `matchPDict` functions, etc...) or for finding palindromes or complemented palindromes in a reference sequence (`findPalindromes` and `findComplementedPalindromes` functions).
- String alignment functions (`pairwiseAlignment` function and related utilities). Not covered in this lab.

2.2 BSgenome data packages

The Bioconductor project also provides a collection of *BSgenome data packages* (e.g. BSgenome.Dmelanogaster.UCSC.dm3). Each of them contains the full genomic sequence for a given organism. These sequences are stored in the containers defined in the Biostrings package so they are ready to be used (no format conversion needed). Because of this, these packages are also called *Biostrings-based genome data packages*. Regardless to the particular genome that they contain, all the *BSgenome data packages* packages are very similar and can be manipulated in a consistent and easy way. They all require the BSgenome software package in order to work properly.

Note that these packages are big (from a few Mb to more than 800 Mb for BSgenome.Hsapiens.UCSC.hg18) so installating them over the network can take a long time.

3 Checking your installation

Exercise 1

1. *Start R and load the Biostrings package.*
2. *Then load the BSgenome package and use the* `available.genomes` *function to check the list of BSgenome data packages that are available on the Bioconductor repositories for your version of R.*
3. *Check that you have BSgenome.Dmelanogaster.UCSC.dm3 (try to load it).*

4 Basic containers

4.1 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. *Use the* `DNAString` *constructor to convert R character string* `"aagtac"` *into a DNAString instance.*
2. *Use* `nchar` *and* `alphabetFrequency` *on it.*
3. *Get its reverse complement with* `reverseComplement`*.*
4. *Extract an arbitrary substring with* `subseq`*.*

4.2 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 sequences stored in a *DNAStringSet* object and the subsetting operator `[` to subset it. In addition, the subsetting operator `[[` can be used to extract an arbitrary element as a *DNAString* object.

Exercise 3

1. *Use the* `DNAStringSet` *constructor to convert R character vector* `c("aagtac", "ccc", "gtt")` *into a DNAStringSet instance. Let's call this instance* `x0`*.*
2. *Use* `width` *and* `length` *on* `x0`*.*

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

4. *How would you invert the order of its elements?*

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

6. *Use the* `DNAStringSet` *constructor to remove the last 2 nucleotides of each element.*

7. *Try* `alphabetFrequency` *and* `reverseComplement` *on* `x0`*.*

For the next exercise, you need the `hgu95av2probe` package. It contains all the probe sequences for microarray hgu95av2 from Affymetrix.

Exercise 4

1. *Load the* `hgu95av2probe` *package.*

2. *Store the* `sequence` *variable of the* `hgu95av2probe` *object into a DNAStringSet object (let's call this object* `x1`*). Display it.*

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

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

4.3 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 available for *XStringViews* objects. Additional `subject`, `start`, `end` and `gaps` methods are also provided.

Exercise 5

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.*

5 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 6

1. *Load BSgenome.Celegans.UCSC.ce2 and display its top level object.*
2. *Display some of the chromosomes. Do they contain IUPAC extended letters?*
3. *Load BSgenome.Dmelanogaster.UCSC.dm3 (Fruit Fly genome). Do the chromosomes contain IUPAC extended letters?*
4. *Put the first 5M bases from Fruit Fly chromosome 2L into a* `DNAString` *object and slice it, that is, create the set of views obtained by moving an imaginary windows of width 20 from the 5' end to the 3' end of the sequence and one nucleotide at a time.*
5. *Use* `chartr` *to simulate a bisulfite transformation of chromosome 4.*

Starting with Bioconductor 2.2 (the current release), some *BSgenome data packages* contain *masked sequences*, that is, chromosome sequences that have built-in masks defined on them (those masks are not active by default). More on this later...

6 String matching

6.1 The matchPattern function

This function finds all matches of a given pattern in a reference sequence called *the subject*.

Exercise 7

1. *Find all the matches of an arbitrary nucleotide sequence in Fruit Fly chromosome 2L. (Don't choose it too short!)*

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.)*

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 Fruit Fly chromosome 2L.*
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.*

6.2 Masking

Starting with Bioconductor 2.2, some *BSgenome data packages* contain *masked sequences*, that is, chromosome sequences that have built-in masks defined on them. There are 3 built-in masks per sequence:

1. the mask of assembly gaps,
2. the mask of repeat regions (as reported by RepeatMasker),
3. and the mask of tandem repeats with a period <= 12 (as reported by Tandem Repeats Finder).

Note that each of these masks corresponds to a *track* in the UCSC genome browser.

Each masked sequence is stored in a new container: the *MaskedDNAString* class. Use the `unmasked` accessor to turn a *MaskedDNAString* object into a *DNAString* object (the mask information will be dropped). Use the `masks` accessor to extract the mask information (the sequence that is masked will be dropped). The built-in masks are NOT active by default. They can be activated individually with:

```
> chr1 <- Hsapiens$chr1
> active(masks(chr1))[3] <- 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. *Activate the mask of assembly gaps in Human chromosome 1. What percentage of the sequence is masked?*
2. *Check the alphabet frequency of this masked sequence (compare with non-masked chromosome 1).*
3. *Activate all masks in Human chromosome 1 and find the occurences of an arbitrary DNA pattern in it. Compare to what you get with non-masked 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).

Exercise 10

1. *Activate the mask of assembly gaps in Human chromosome 2 and check its alphabet frequency. Do you see something unexpected?*
2. *Use* `as(chr2 , "XStringViews")` *to turn this masked sequence into an XStringViews object. Do you see the unmasked Ns?*
3. *Which contig do they belong to?*
4. *Have a look at the inter-view gaps of this XStringViews object.*

6.3 Finding palindromes

The `findPalindromes` function finds palindromic regions in a sequence. The palindromes that will be detected are either strict palindromes or 2-arm palindromes i.e. palindromes where the 2 arms are separated by an arbitrary sequence called "the loop". The `findComplementedPalindromes` function finds complemented palindromes i.e. palindromes where the 2 arms are reverse-complementary sequences from each other.

Exercise 11

1. *Find all the complemented palindromes with arms of at least 30 bp and a loop of at most 10 bp in Fruit Fly chromosome X.*
2. *Try to do the same with Human chromosome 2. Which palindromic regions contain the 4 bases?*

Note that `findPalindromes` and `findComplementedPalindromes` are still a work-in-progress: they don't support the `max.looplength` and `max.mismatch` args yet and they don't return a "clean" set of palindromic regions (the same palindromic region is reported several times). These issues need to be addressed in future versions of Biostrings.

6.4 Finding the hits of a large set of short motifs

The set of short reads that one gets from an ultra-high throughput sequencing experiments can be very large (typically a few millions of 35-mers with a Solexa machine) so we need fast algorithms if we want to be able to *align* all those reads to the reference genome in reasonable time.

Starting with Bioconductor 2.2, this can be done with the `matchPDict` function from the Biostrings package. Note that this new functionality is currently under active development in the devel version of Bioconductor (2.3) that will be released in the fall of 2008. If you are planning to use Bioconductor for processing ultra-high throughput sequencing data, you are invited to check the ShortRead package (not released yet, available only in BioC devel) and to subscribe to the bioc-sig-sequencing mailing list (see http://bioconductor.org/docs/mailList.html for the details).

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 12
In this exercise we want to find the hits of all the probes of chip hgu95av2 in Human chromosome 1. For now, "hit" means "location of an exact match".

1. *Load the BSgenome data package for Human (hg18 from UCSC).*
2. *Load the hgu95av2probe package and put the probe sequences in a `DNAStringSet` object (let's call this object `dict0`).*
3. *Preprocess `dict0` with the `PDict` constructor.*
4. *Call `matchPDict` on this PDict object (1st arg) and Human chromosome 1 (2nd arg). Put the result in a variable called `mindex` (`matchPDict` returns an MIndex object).*
5. *Use `countIndex` on `mindex` to extract the count of hits.*
6. *Which probe has the highest number of hits? Display those hits as an XStringViews object.*

7 SNP injection

When the patterns to match against a genome of reference come from a sequencing experiment then they will contain the SNPs of the individual that was used for the experiment. In this case looking for exact matches will miss a lot of hits, not only because of the sequencing errors inherent to the sequencing technology, but because of the presence of SNPs.

This second type of hit loss can be partially avoided by *injecting* the known SNPs in the reference genome, that is, by replacing the non-ambiguous letter by an IUPAC ambiguity letter at each SNP location in the reference genome. Then, when `matchPattern` or `matchPDict` are used on this modified genome and with `fixed=FALSE`, hits that contain a known SNP will be found too. This provides a way to detect known SNPs in the individual.

Starting with Bioconductor 2.2, one *SNPlocs* package is available: the SNPlocs.Hsapiens.dbSNP.20071016 package. It contains the locations of all known SNPs for Human (extracted from dbSNP) together with the alleles information (an IUPAC ambiguity letter for each SNP). It is aimed to be used in conjonction with the BSgenome.Hsapiens.UCSC.hg18 package (more on this below).

Note that the *SNPlocs* packages are a new type of annotation packages and more packages can be added in the future if they are found useful. The `available.SNPs` function (from the BSgenome software package) will allow you to check the list of *SNPlocs packages* that are available on the Bioconductor repositories for your version of R.

It's easy to *inject* SNPs in the reference genome. For example, with the Human genome:

```
> library(SNPlocs.Hsapiens.dbSNP.20071016)
> hg18snp <- injectSNPs(Hsapiens, 'SNPlocs.Hsapiens.dbSNP.20071016')
```

The resulting `hg18snp` object is a modified version of the original genome (the `Hsapiens` object) where IUPAC ambiguity letters have been injected in the chromosome sequences at the SNP locations. You can display and use `hg18snp` exactly in the same way that you use `Hsapiens` (both are *BSgenome* objects).

Exercise 13
In this exercise, we reuse the PDict object obtained by preprocessing the probe sequences of the hgu95av2probe package.

1. *Inject the SNPs from SNPlocs.Hsapiens.dbSNP.20071016 in the hg18 genome and display the resulting BSgenome object (note the additional line `"with SNPs injected from..."`).*

2. *Load the modified sequence for chromosome 1 and look at its alphabet frequency (compare with the original chromosome 1).*

3. *Mask the assembly gaps in this modified chromosome 1 and look at its alphabet frequency again.*

4. *Use* `countPDict` *with* `fixed="pattern"` *to count the nb of hits for all the hgu95av2 probes.*

5. *Try to "see" some hits with SNPs.*

8 Genome-wide analysis

So far you've written code for performing single chromosome analyses. Most of the time, your analysis will need to be extended to the entire genome (plus and minus strands of all chromosomes) so your code will need to be adapted.

For example here is some simple code that finds all the hits of the drosophila2 probes (Affymetrix microarray) in the Fruit Fly genome:

```
> walkGenome <- function(bsgenome, seqnames, pdict, objname_suffix)
+ {
+     for (seqname in seqnames) {
+         objname <- paste(seqname, objname_suffix, sep="")
+         filename <- paste(objname, ".rda", sep="")
+         cat("Walking ", seqname, " (will save hits ",
+             "in ", filename, " file)... ", sep="")
+         subject <- unmasked(bsgenome[[seqname]])
+         hits <- matchPDict(pdict, subject)
+         rm(subject)
+         unload(bsgenome, seqname)
+         assign(objname, hits)
+         save(list=objname, file=filename)
+         rm(hits, list=objname)
+         cat("OK\n")
+     }
+ }
> library(drosophila2probe)
> dict0 <- DNAStringSet(drosophila2probe$sequence)
> library(BSgenome.Dmelanogaster.UCSC.dm3)
> SEQNAMES <- seqnames(Dmelanogaster)
> ### Find hits in the + strands
> pdict <- PDict(dict0)
> walkGenome(Dmelanogaster, SEQNAMES, pdict, "phits")
> ### Find hits in the - strands
> rc_dict0 <- reverseComplement(dict0)
> pdict <- PDict(rc_dict0)
> walkGenome(Dmelanogaster, SEQNAMES, pdict, "mhits")
```

Writing code that performs genome-wide analysis is described in the Genome-Searching vignette from the BSgenome software package. Note that this vignette is still a work-in-progress (feedback is always appreciated).