Biostrings

José Reyes

Centro de Ciencias Genómicas
Universidad Nacional Autónoma de México

August 29, 2009
Outline

What is a Biostring?

Sources of biological sequences

Exploring a sequence

Pattern matching
Introduction I

- Bioinformatics is focus on the analysis of the informational molecules that give origin to living organisms.

- What aspects of these sequences can make limit our ability to analyze them?
The Biostrings package was created to provide an efficient way for representing and analyzing these sequences.

There are three main types of Biostrings:

- DNAString
- RNAString
- AAString
Packages for this session I

> library(Biostrings)
> library(BSgenome)
> library(biomaRt)
> library(GenomeGraphs)
Creating a random string

To begin the session, we need to create a DNA sequence. How would you generate a random DNA string in R?
Creating a random string

> DNA_ALPHABET

[1] "A" "C" "G" "T" "M" "R" "W" "S" "Y"
[10] "K" "V" "H" "D" "B" "N" "-" "+"

> seq <- sample(DNA_ALPHABET[1:4],
+ size = 24, replace = TRUE)
> seq <- DNASTring(paste(seq, collapse = ""))
> seq

24-letter "DNASTring" instance
seq: TTAGTTTCACATAGCCAGCAGCCTGCC
Basic Biostrings functions I

- `alphabetFrequency`

  ```r
  alphabetFrequency(seq, baseOnly = T, +
  as.prob = T)
  ```

  ```
  A     C     G     T
  0.2083333  0.3333333  0.1666667  0.2916667
  other
  0.0000000
  ```

- `reverseComplement`

  ```r
  reverseComplement(seq)
  ```

  ```
  24-letter "DNASTRING" instance
  seq: GGCAGGCTGGCTATGTGAAACTAA
  ```

- `translate`

  ```r
  translate(seq)
  ```

  ```
  8-letter "AASTRING" instance
  seq: LVSHSQPA
  ```
Obtaining a subsequence I

You can access certain positions by using normal subset operators:

```r
> seq[3:10]
```

8-letter "DNAString" instance

```
seq: AGTTTCAC
```

However, Biostrings provide the `subseq` function. This function follows the SEW interface, meaning that the subsequence can be defined by two out of three possible parameters:

- start
- end
- width
Obtaining a subsequence

```r
> subseq(seq, start = 3, end = 10)

  8-letter "DNAString" instance
seq: AGTTTCAC

> subseq(seq, start = 3, width = 8)

  8-letter "DNAString" instance
seq: AGTTTCAC

> subseq(seq, end = 10, width = 8)

  8-letter "DNAString" instance
seq: AGTTTCAC
```
Obtaining a subsequence I

This function is very versatile. It even allows negative positions:

> subseq(seq, start = 1, end = -4)

21-letter "DNAString" instance
seq: TTAGTTTCACATAGCCAGCCT

What does a negative position mean?
Collections of Biostrings I

* The Biostrings package also provides another type of object, named **XStringSet** (The X can stand for DNA, RNA or AA).

* Let’s create a **DNASTringSet** object:

```
> set <- NULL
> for (i in c(1:4)) set <- c(set, +   paste(sample(DNA_ALPHABET[1:4], +     30, replace = T), collapse = ""))
> set

[1] "CATGCAAATACCTTTTATTTGGGGGTCAGAA"
[2] "GCTAAGCGGATTGGAGCCCTCCTCCTTTTAG"
[3] "CAACCCGCATGGTAAGTTGACACCACCCGT"
[4] "TACCTTGGGTTACCCCGCGCAGCTTGCTCT"

> set <- DNASTringSet(set)
> set
```
A DNAStringSet instance of length 4

<table>
<thead>
<tr>
<th>width</th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>CATGCAAAATACCTTTTATTGGGGGTCAAGAA</td>
</tr>
<tr>
<td>30</td>
<td>GCTAAGCGGATTGGAGCCCTCCTCCTTTTAG</td>
</tr>
<tr>
<td>30</td>
<td>CAACCCGCATGGTAAGTTGACACCACCCCGT</td>
</tr>
<tr>
<td>30</td>
<td>TACCTTGGGTACCCCCCGGCAGCTTGCTCT</td>
</tr>
</tbody>
</table>
Collections of Biostrings I

- You can use the `reverseComplement`, `alphabetFrequency` and `subseq` over all the Biostrings in your collection.
- `length` now returns the number of sequences, and `width` returns the length of each sequence.
- An useful thing is that you can put `names` to the sequences.

```r
> names(set) <- seq(4)
> set

A DNAStringSet instance of length 4

<table>
<thead>
<tr>
<th>width</th>
<th>seq</th>
<th>names</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>30 CAT...GAA</td>
<td>1</td>
</tr>
<tr>
<td>[2]</td>
<td>30 GCT...TAG</td>
<td>2</td>
</tr>
<tr>
<td>[3]</td>
<td>30 CAA...CGT</td>
<td>3</td>
</tr>
<tr>
<td>[4]</td>
<td>30 TAC...TCT</td>
<td>4</td>
</tr>
</tbody>
</table>
```
There is a special function for reading FASTA files and creating a XStringSet: `read.DNAStringSet` (you can also read proteins by changing the prefix).

The function for writing a FASTA file from an XStringSet is `write.XStringSet`.

Using this function can save you a lot of time in your phylogenies project, for example, by editing the names of your sequences, creating a subfile with just some organisms or editing the alignment to eliminate gaps.
A package that is related to Biostrings is **BSgenome**

BSgenome provides preprocessed genomes from some model organisms, as Biostrings.

```r
> available.genomes()
```

In this session we will use the *Escherichia coli APEC O1* genome (NC_008563), so:

```r
> require(BSgenome.Ecoli.NCBI.20080805)
> eco <- Ecoli$NC_008563
```
Generating views I

- An object of **XStringViews** represents a set of "subsequences" from a subject string that are defined by the **StartEndWidth** interface.
- The views are generated by the function **Views** and can be defined in different ways:

  ```r
  > Views(eco, start = c(10, 20, 30, + 40), end = c(50, 60, 70, 80))
  
  Views on a 5082025-letter DNAString subject
  subject: AACGGGCAATATGT...TTCATTCTGACTGC
  views:
     start end width
  [1] 10 50 41 [TATGTCTC...ATAGCAG]
  [2] 20 60 41 [TGTGGATT...CTGAACT]
  [3] 30 70 41 [AAAAAGAG...TACCTGC]
  [4] 40 80 41 [TCTGATAG...GAGTAAA]
  
  > Views(eco, start = c(10, 20, 30, + 40), end = c(50, 60))
  ```
Views on a 5082025-letter DNAString subject

subject: AACGGGCAATATGT...TTCATTCTGACTGC

views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
<th>view</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
<td>41</td>
<td>[TATGTCTC...ATAGCAG]</td>
</tr>
<tr>
<td>20</td>
<td>60</td>
<td>41</td>
<td>[TGTGGATT...CTGAAC]</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>21</td>
<td>[AAAAAGAG...ATAGCAG]</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
<td>21</td>
<td>[TCTGATAG...CTGAAC]</td>
</tr>
</tbody>
</table>

> Views(eco, start = c(10, 20, 30, + 40), width = c(100))

Views on a 5082025-letter DNAString subject

subject: AACGGGCAATATGT...TTCATTCTGACTGC

views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
<th>view</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>109</td>
<td>100</td>
<td>[TATGTCTC...CACTAA]</td>
</tr>
<tr>
<td>20</td>
<td>119</td>
<td>100</td>
<td>[TGTGGATT...TTTAAC]</td>
</tr>
<tr>
<td>30</td>
<td>129</td>
<td>100</td>
<td>[AAAAAGAG...ATAGGA]</td>
</tr>
<tr>
<td>40</td>
<td>139</td>
<td>100</td>
<td>[TCTGATAG...CGCAC]</td>
</tr>
</tbody>
</table>
Small exercise I

1. Sample the E. coli genome by generating 1000 random views of variable width (50 - 500).
2. Use the alphabetFrequency function over these views
3. Repeat the previous steps, but this time do them with only 100 samples.
4. Use the same function to obtain the composition of the whole chromosome and compare the results.

```r
> v1 <- Views(eco, start = sample(seq(length(eco)),
+   size = 1000, replace = TRUE),
+   width = sample(50:500, size = 1000, 
+     replace = TRUE))
> v2 <- Views(eco, start = sample(seq(length(eco)),
+   size = 100, replace = TRUE),
+   width = sample(50:500, size = 100, 
+     replace = TRUE))
> alphabetFrequency(v1, baseOnly = T, 
+   as.prob = T, collapse = T)
```
Small exercise II

```
A   C   G   T
0.2458811 0.2533246 0.2527131 0.2480812
other
0.0000000

> alphabetFrequency(v2, baseOnly = T, 
+     as.prob = T, collapse = T)

A   C   G   T
0.2432266 0.2593597 0.2518358 0.2455779
other
0.0000000

> alphabetFrequency(eco, baseOnly = T, 
+     as.prob = T)

A   C   G
2.471704e-01 2.529128e-01 2.525602e-01
T   other
2.473315e-01 2.518681e-05
```
The sliding windows I

- Bioinformaticians love to use sliding windows for their analysis. Briefly, sliding windows are overlapping fragments of a sequence, generated by "walking" through it.

- How would you create a set of windows of width = 100, and sliding step = 10, of the first 10kb of E. coli’s genome?
I know two ways, but I’m sure there are more:

```r
> v1 <- Views(eco, start = seq(from = 1, 
+    to = 9901, by = 10), width = 100)
> v2 <- successiveViews(eco, from = 1,
+    width = rep(100, 991), gapwidth = -90)
> head(v1)

Views on a 5082025-letter DNAString subject
subject: AACGGGCAATATGT...TTCATTCTGACTGC
views:
    start  end  width
[1]    1   100  100  [AACGGGCA...GACTTAG]
[2]   11   110  100  [ATGTCTCT...ACTAAAT]
[3]   21   120  100  [GTGGATTA...TTAACCA]
[4]   31   140  100  [AAAAGAGT...TAGGCAT]
[5]   41   150  100  [CTTCTGAA...ATAAAAA]
> tail(v2)
```

> tail(v2)

```r
```
Views on a 5082025-letter DNAMString subject
subject: AACGGGCAATATGT...TTCATTCTGACTGC
views:

<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9851</td>
<td>9950</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>9861</td>
<td>9960</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>9871</td>
<td>9970</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>9881</td>
<td>9980</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>9891</td>
<td>9990</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>9901</td>
<td>10000</td>
<td>100</td>
</tr>
</tbody>
</table>

What kind of analysis do you think you can make with an approach like this one?
Last but not least I

- Biostrings provide useful pattern matching functions:
  - `matchPattern`: For matching one pattern to one string.
  - `vmatchPattern`: For matching one pattern to several strings (StringSet).
  - `matchPDict`: For matching a dictionary of equal length patterns to a string.
  - `vmatchPDict`: For matching a dictionary of patterns to a collection of strings.

- Check the help of Biostrings and look for other interesting pattern matching functions.
Exercise I

How many restriction fragments do you expect to have if you digest the first 10kb of E. coli genome with EcoR1 (GAATTC)?
matchPattern I

> motif <- DNAString("GAATTC")
> tail(matchPattern(motif, eco))

Views on a 5082025-letter DNAString subject
subject: AACGGGCAATATGT...TTCATTCTGACTGC
views:

```
  start   end  width
  [1] 5012393 5012398  6 [GAATTC]
  [2] 5047471 5047476  6 [GAATTC]
  [3] 5056207 5056212  6 [GAATTC]
  [4] 5056677 5056682  6 [GAATTC]
  [5] 5068417 5068422  6 [GAATTC]
  [6] 5075296 5075301  6 [GAATTC]
```
Two restriction enzymes I

What would you do if you wanted to digest also with BamH1 (GGATCC)?
> m1 <- DNAStringSet("GAATTC")
> m2 <- DNAStringSet("GGATCC")
> dict <- PDict(append(m1, m2))
> restrict <- matchPDict(dict, eco)
> restrict

MIndex object of length 2

> tail(restrict[[1]])

IRanges instance:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>5012393</td>
<td>5012398</td>
<td>6</td>
</tr>
<tr>
<td>5047471</td>
<td>5047476</td>
<td>6</td>
</tr>
<tr>
<td>5056207</td>
<td>5056212</td>
<td>6</td>
</tr>
<tr>
<td>5056677</td>
<td>5056682</td>
<td>6</td>
</tr>
<tr>
<td>5068417</td>
<td>5068422</td>
<td>6</td>
</tr>
<tr>
<td>5075296</td>
<td>5075301</td>
<td>6</td>
</tr>
</tbody>
</table>
This is the end of the lecture. You can practice some of the functions I just told you about with some exercises.