Biostrings

José Reyes

Centro de Ciencias Genómicas
Universidad Nacional Autónoma de México

August 29, 2009

Outline

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?

Introduction I

- 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 I

```
> 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: TTAGTTTCACATAGCCAGCCTGCC
```

Basic Biostrings functions I

- alphabetFrequency

```
> alphabetFrequency(seq, baseOnly = T,
+     as.prob = T)
        A         C         G         T
0.2083333 0.3333333 0.1666667 0.2916667
    other
0.0000000
```

- reverseComplement

```
> reverseComplement(seq)
  24-letter "DNAString" instance
seq: GGCAGGCTGGCTATGTGAAACTAA
```

- translate

```
> translate(seq)
  8-letter "AAString" instance
seq: LVSHSQPA
```

Obtaining a subsequence I

You can access certain positions by using normal subset operators:

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

```
  8-letter "DNAString" instance
seq: AGTTTCAC
```

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

- start
- end
- width

Obtaining a subsequence I

```
> 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 if 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] "CATGCAAATACCTTTTATTGGGGGTCAGAA"
[2] "GCTAAGCGGATTGGAGCCCTCCTCCTTTAG"
[3] "CAACCCGCATGGTAAGTTGACACCACCCGT"
[4] "TACCTTGGGTTACCCCGCGCAGCTTGCTCT"
```

```
> set <- DNAStringSet(set)
> set
```

Collections of Biostrings II

```
  A DNAStringSet instance of length 4
    width seq
[1]    30 CATGCAAATACCTTTTATTGGGGGTCAGAA
[2]    30 GCTAAGCGGATTGGAGCCCTCCTCCTTTAG
[3]    30 CAACCCGCATGGTAAGTTGACACCACCCGT
[4]    30 TACCTTGGGTTACCCCGCGCAGCTTGCTCT
```

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.

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

```
  A DNAStringSet instance of length 4
    width seq         names
[1]    30 CAT...GAA 1
[2]    30 GCT...TAG 2
[3]    30 CAA...CGT 3
[4]    30 TAC...TCT 4
```

Reading FASTA files I

- 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.

Preprocessed genomes I

- A package that is related to Biostrings is BSgenome
- BSgenome provides preprocessed genomes from some model organisms, as Biostrings.

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

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

```
> 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:

```
> 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))
```

Generating views II

```
  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  50    21 [AAAAAGAG...ATAGCAG]
[4]    40  60    21 [TCTGATAG...CTGAACT]
```

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

```
  Views on a 5082025-letter DNAString subject
subject: AACGGGCAATATGT...TTCATTCTGACTGC
views:
    start end width
[1]    10 109   100 [TATGTCTC...CACTAAA]
[2]    20 119   100 [TGTGGATT...TTTAACC]
[3]    30 129   100 [AAAAAGAG...ATAGGCA]
[4]    40 139   100 [TCTGATAG...CGCACAG]
```

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.

```
> 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?

The sliding windows I

I know two ways, but I'm sure there are more:

```
> 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]    31 130   100 [AAAAGAGT...TAGGCAT]
[5]    41 140   100 [CTGATAGC...GCACAGA]
[6]    51 150   100 [CTTCTGAA...ATAAAAA]
```

```
> tail(v2)
```

The sliding windows II

```
  Views on a 5082025-letter DNAString subject
subject: AACGGGCAATATGT...TTCATTCTGACTGC
views:
    start   end width
[1]  9851  9950   100 [TATATTG...GAGCGG]
[2]  9861  9960   100 [TTGCACG...AGCTTA]
[3]  9871  9970   100 [TTGTAGG...TTAGTG]
[4]  9881  9980   100 [GATAAGG...TCACCA]
[5]  9891  9990   100 [TCACGCC...GCAGAA]
[6]  9901 10000   100 [TCCGGCA...GCGACC]
```

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)?

matchPDict I

```
> 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:
      start     end width
[1] 5012393 5012398     6
[2] 5047471 5047476     6
[3] 5056207 5056212     6
[4] 5056677 5056682     6
[5] 5068417 5068422     6
[6] 5075296 5075301     6
```

The end I

This is the end of the lecture. You can practice some of the functions I just told you about with some exercises.