GeneR

JORGE ARTURO ZEPEDA MARTINEZ
LOPEZ HERNANDEZ JOSE FABRICIO.
jzepeda@lcg.unam.mx
jlopez@lcg.unam.mx

October 6, 2009

Abstract

GeneR packages allow direct use of nucleotide sequences within R software. Functions can be used to read and write sequences from main file formats (Embl, Genbank and Fasta) in order to perform a lot of manipulations and analyses. Main functions are implemented with C extensions.

For this package, the functions are in 4 categories:
1.Reading and writing sequences.
2.Handling sequences.
3.Analyzing sequences.
4.Genetic tools.

INSTALLATION.

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("GeneR")

Using R version 2.10.0 (R-devel), biocinstall version 2.5.5.
Installing Bioconductor version 2.5 packages:
[1] "GeneR"
Please wait...

> library("GeneR")
```

1 Basic manipulation with sequences as character string Standard R commands allow to extract subpart of a character string, or append two character strings, put in upper case (functions substr, paste, toupper). GeneR also has tools for the specic use of genetic sequences: insert a sequence into a another, compute the reverse complementary, count mono, di or tri-nucleotides of sequence and the possibility to import sequence from a Embl, Fastaor GeneBank file.

Examples:

Insert a poly A into sequence

```
> seq <- "gtcatgcatgctaggtgacagttaaaatgcgtctaggtgacagtctaacaa"
> seq2 <- insertSeq(seq, "aaaaaaaaaaaaaa", 20)
> seq2
```

```
[1] "gtcatgcatgctaggtgacaaaaaaaaaaaaaaagttaaaatgcgtctaggtgacagtctaacaa"
```

Compute the reverse complementary:

```
> strComp(seq)

[1] "ttgttagactgtcacctagacgcattttaactgtcacctagcatgcatgac"
```

Count mono nucleotides, di-nucleotides (wsize can be from 1 to 15 if computer has enough memory):

```
> strCompoSeq(seq2, wsize = 1)

       T         C         A   G X
[1,] 0.2 0.1384615 0.4615385 0.2 0

> strCompoSeq(seq2, wsize = 2)

     TT TC      TA     TG TX      CT CC      CA CG CX      AT AC      AA     AG
[1,]  0  0 0.03125 0.0625  0 0.09375  0 0.15625  0  0 0.03125  0 0.28125 0.0625
     AX     GT      GC     GA GG GX XT XC XA XG XX
[1,]  0 0.1875 0.03125 0.0625  0  0  0  0  0  0  0
```

Get some data from the web:

```
> seqNcbi("BY608190", file = "arch.fa")

[1] 1

> strReadFasta("arch.fa", from = 10, to = 35)

[1] "TGCGTTTGTTTTTTAGTGACTTCTAC"
```

2 Manipulation of sequences with buffers. The data is loaded into the buffer and all computation done with GeneR functions, only usefull results returns to standard R objects. Which is very useful specially if you are working with great amount of data such as an entire chromosome.

Examples:

Sequence read from a fasta file:

```
> readFasta("arch.fa")

[1] 0

> readFasta("arch.fa", seqno = 1)

[1] 0
```

Show current buffer:

```
> getSeq(from = 10, to = 35)

[1] "TGCGTTTGTTTTTTAGTGACTTCTAC"
```

Sames previous functions with buffers: Extract from multiple positions.

```
> getSeq(seqno = 0, from = c(1, 10, 360), to = c(10, 20, 0))

[1] "TGGGCTTATT"                  "TGCGTTTGTTT"
[3] "AATAAAGCCAACTTGCAGCTGCTGTT"

> assemble(seqno = 0, destSeqno = 1, from = c(1, 10, 360), to = c(10,
+     20, 0))

[1] 1

> getSeq(seqno = 1)

[1] "TGGGCTTATTTGCGTTTGTTTAATAAAGCCAACTTGCAGCTGCTGTT"
```

See the length of a sequence:

```
> sizeSeq(seqno = 0)

[1] 385

> size0 <- sizeSeq(seqno = 0)
> getSeq(seqno = 0, from = size0 - 20, to = 0)

[1] "AGCCAACTTGCAGCTGCTGTT"
```

Append and concat:

```
> getSeq(0, from = 1, to = 35)

[1] "TGGGCTTATTGCGTTTGTTTTTTAGTGACTTCTAC"

> getSeq(1)

[1] "TGGGCTTATTTGCGTTTGTTTAATAAAGCCAACTTGCAGCTGCTGTT"

> appendSeq(0, 1)

[1] 0

> getSeq(seqno = 0, from = size0 - 20, to = size0 + 10)

[1] "AGCCAACTTGCAGCTGCTGTTTGGGCTTATT"

> concat(seqno1 = 0, seqno2 = 1, destSeqno = 3, from1 = 2, to1 = 10,
+     from2 = 8, to2 = 0, strand1 = 1)

[1] 3

> getSeq(3)

[1] "AATAAGCCCAACAGCAGCTGCAAGTTGGCTTTATTAAACAAACGCAAAT"
```

3 Look for motifs, composition, masked position:
Look for motives

```
> exactWord("AAAG", seqno = 0)
```

```
[[1]]
[1] 234 363 410
```

Find Orfs (returns positions of predicted ORFs start position = start codon, stop position = stop codon)

```
> getOrfs(seqno = 0)

start stop
[1,] 36 74
[2,] 42 74
[3,] 49 96
[4,] 53 58
[5,] 159 254
[6,] 175 186
[7,] 198 254
[8,] 240 254

> maxOrf(seqno = 0)

[1] 96
```

Mask sequences while lowering case of parts of sequence:

```
> x = c(5, 15, 30)
> y = c(10, 20, 35)
> lowerSeq(from = x, to = y)

[1] 1

> getSeq(from = 1, to = 35)

[1] "TGGGcttattGCGTttgtttTTTAGTGACttctac"
```

And the reverse: get masked positions from a sequence:

```
> posMaskedSeq(seqno = 0, type = "lower")

     from to
[1,]    5 10
[2,]   15 20
[3,]   30 35

> posMaskedSeq(seqno = 0, type = "upper")

     [,1] [,2]
[1,]    1    4
[2,]   11   14
[3,]   21   29
[4,]   36  432
```

Change from Dna to Rna alphabet:

```
> dnaToRna()
```

```
[1] 0
```

```
> getSeq()
```

```
[1] "UGGGcuuauuGCGUuuguuuUUUAGUGACuucuacAUGCUGAUGUCCCAUGUAUGUAGUCUUAGACCUGUUUAAUA
UCUGUAACUAUCAGCUAUAAUAUUGUGGUGACCACNUGCUAUAGGAUUUUGCCUCUGUGUUAACUACAACAUAUUG
GAUUGUAUGUGUCUGUGCCUCCAUGGGGGAUUAGAGCCAGAGGGAAUGUCUUCUUUGCUCUGUUUCUUUUAUAUUU
AUAACAAAGAUAUGGAUACUUUCUAGUGAAUGCAUAAGUUAGUGUGCUUUUCUUAUUUUGCUUUAAAUUUCAAGUU
UUUACACUGCUGUGAUAAAAUCCUCACAGAUAGCAUCCUCUGGAUGGCACAGUACAAUAAAGCCAACUUGCAGCUG
CUGUUUGGGCUUAUUUGCGUUUGUUUAAUAAAGCCAACUUGCAGCUGCUGUU"
```

4 Adresses manipulations: GeneR developeres have defined 3 types of addresses on a subsequence extracted from a master sequence:
-Absolute addresses i.e. addresses on the master sequence, from the 5' end of the input strand refered as forward (noted A)
-Real addresses, i.e. addresses on the master sequence, from the 5' end of one of strands (noted R)
-Relative addresses, i.e. addresses on working subsequence, from the 5' end of one of strands (noted T)

Although all functions using positions need and return absolute addresses, 6 functions allow to convert R, A, T into any other type (functions RtoA, RtoT, AtoR, AtoT, TtoR, TtoA).

For example lets imagine that we work with a certain chromosome, and we study only part of chromosome between position 11 to 25:

```
> seq <- "xxxxxxxxxxATGTGTCGTTAATTGxxxxxxxxxxxxxxx"
> placeString(seq)
```

```
[1] 0
```

```
> setStrand(0)
```

```
[1] 0
```

```
> writeFasta(file = "arch.fa")
```

```
[1] 1
```

```
> readFasta(file = "arch.fa", from = 11, to = 25)
```

```
[1] 0
```

```
> getSeq()
```

```
[1] "ATGTGTCGTTAATTG"
```

Orfs on 'Absolute' and 'relative' Adresses:

```
> getOrfs()
```

```
start stop
[1,] 11 22
```

```
> AtoT(getOrfs())

start stop
[1,] 1 12
```

More Fun:

```
> getSeq()

[1] "ATGTGTCGTTAATTG"

> exactWord(word = "AA")

[[1]]
[1] 21

> AtoT(exactWord(word = "AA")[[1]])

start stop
[1] 11
```

5 Bank files: -Fasta files

There are 2 functions to get sequence files from internet: seqUrl (from a srs server) and seqNcbi (from Ncbi). Sequences are retrieved in Fasta or Embl format for seqUrl and Fasta or GenBank format for seqNcib.

```
> seqNcbi("BY608190", file = "bank.fa")

[1] 1

> seqNcbi("BY608190", file = "BY608190.gbk", type = "genbank")

[1] 1
```

Basic tools to get or write informations from bank file

```
> readFasta(file = "bank.fa", name = "gi|26943372|gb|BY608190.1|BY608190",
+     from = 1, to = 30)

[1] 0

> getSeq()

[1] "TGGGCTTATTGCGTTTGTTTTTTAGTGACT"

> fastaDescription(file = "bank.fa", name = "gi|26943372|gb|BY608190.1|BY608190")

[1] "BY608190 RIKEN full-length enriched, visual cortex Mus musculus cDNA
clone K230301C17 3', mRNA sequence"
```

And to write a to the fasta bank file:

```
> s = "cgtagctagctagctagctagctagcta"
> placeString(s, seqno = 0)
```

```
[1] 0
```

```
> writeFasta(seqno = 0, file = "bank.fa", name = "MySequence",
+     comment = "A sequence Generated by R", append = TRUE)
```

```
[1] 1
```

Show bank file

```
> cat(paste(readLines("bank.fa"), collapse = "\n"))
```

```
>gi|26943372|gb|BY608190.1|BY608190 BY608190 RIKEN full-length enriched, visual cortex
Mus musculus cDNA clone K230301C17 3', mRNA sequence
TGGGCTTATTGCGTTTGTTTTTTAGTGACTTCTACATGCTGATGTCCCATGTATGTAGTCTTAGACCTGT
TTAATATCTGTAACTATCAGCTATAATATTGTGGTGACCACNTGCTATAGGATTTTGCCTCTGTGTTAAC
TACAACATATTGGATTGTATGTGTCTGTGCCTCCATGGGGGATTAGAGCCAGAGGGAATGTCTTCTTTGC
TCTGTTTCTTTTATATTTATAACAAAGATATGGATACTTTCTAGTGAATGCATAAGTTAGTGTGCTTTTC
TTATTTTGCTTTAAATTTCAAGTTTTTACACTGCTGTGATAAAATCCTCACAGATAGCATCCTCTGGATG
GCACAGTACAATAAAGCCAACTTGCAGCTGCTGTT
>MySequence A sequence Generated by R
CGTAGCTAGCTAGCTAGCTAGCTAGCTAGCTA
```

-Embl files Some basic tools to get sequence and write Embl files

```
> s <- "gtcatgcatcctaggtgtcagggaaaatgcgtctacgtgacagtctaacaa"
> placeString(s)
```

```
[1] 0
```

Add a line with "FT CDS bla bla bla"

```
> writeEmblLine(file = "arch.embl", code = "FT", header = "CDS",
+     text = "<1..12", nextfield = FALSE)
```

```
[1] 1
```

Add lines with "FT bla bla bla"

```
> writeEmblLine(file = "arch.embl", code = "FT", header = "", text = "/codon_start=2",
+     nextfield = FALSE)
```

```
[1] 1
```

```
> writeEmblLine(file = "arch.embl", code = "FT", header = "", text = "/gene=\"toto\"",
+     nextfield = FALSE)
```

```
[1] 1
```

Add sequence

```
> writeEmblSeq(file = "arch.embl")
```

Show file

```
> cat(paste(readLines("arch.embl"), collapse = "\n"))
```

```
FT   CDS             <1..12
FT                   /codon_start=2
FT                   /gene="toto"
SQ   Sequence 51 BP; 15 A; 11 C; 13 G; 12 T; 0 other;
     gtcatgcatc ctaggtgtca gggaaaatgc gtctacgtga cagtctaaca a                51
//
```

-GeneBank files There is a function to open any sequence from a GeneBank files

```
> readGbk(file = "BY608190.gbk", name = "BY608190", from = 10,
+     to = 30)

[1] 0

> getSeq()

[1] "TGCGTTTGTTTTTTAGTGACT"
```

If you want to play with differents RandomSequence, there is a tool to do it.

The sequence could be make by one, two or three letters, using probabilities or numbers of elements for each one.
Examples:

```
> randomSeq(prob = c(0.2, 0.3, 0.2, 0.3), letters = c("T", "C",
+     "A", "G"), n = 30)

[1] "GTTGTCCTCGATGGCACCGGCAGGGGAATA"

> randomSeq(prob = rep(0.0625, 16), letters = c("TT", "TC", "TA",
+     "TG", "CT", "CC", "CA", "CG", "AT", "AC", "AA", "AG", "GT",
+     "GC", "GA", "GG"), n = 10)

[1] "ATTTTTGTCAGGAAACGGTC"

> shuffleSeq(count = c(7, 3, 3, 4, 0), letters = c("T", "C", "A",
+     "G", "N"))

[1] "GATCTGTCGCGATATTT"

> shuffleSeq(count = c(rep(4, 4), rep(2, 4), rep(1, 4), rep(0,
+     4)), letters = c("TT", "TC", "TA", "TG", "CT", "CC", "CA",
+     "CG", "AT", "AC", "AA", "AG", "GT", "GC", "GA", "GG"))

[1] "CATGCGTATGTACTCCTCCTTGCCTTTTTTAATCATTGTATTTCAGTCCACGTAAC"
```

Computes profile of user defined quantities around sites of interest in sequence fragments. A Profile is constituted of bins of equal size around the sites of interest named origins. It produces for each bin, and for each quantity the mean, the standard deviation and the number of valid events.
Example:

```
> s <- ""
> for (i in 1:20) s <- paste(s, randomSeq(n = 100), randomSeq(prob = c(0.3,
+     1, 1, 1, 0)/3.3, n = 100), sep = "")
> placeString(s, seqno = 0)
```

```
[1] 0
```

```
> dens <- densityProfile(ori = 200 * (1:20) - 100, from = 200 *
+     (0:19) + 1, to = 200 * (1:20), seqno = 0, fun = seqSkew,
+     nbinL = 15, nbinR = 15, sizeBin = 5)
> plot(dens$skta, main = "TA skew")
```

```
[1] 1
```