

# Searching a DNA sequence using the `matchPattern` method (work in progress)

Hervé Pagès

August 14, 2007

## Contents

<b>1</b>	<b>Load a genome</b>	<b>1</b>
<b>2</b>	<b>Find patterns in a DNA sequence</b>	<b>2</b>
<b>3</b>	<b>A note on performance</b>	<b>4</b>

## 1 Load a genome

Load the *Caenorhabditis elegans* genome:

```
> library(BSgenome.Celegans.UCSC.ce2)
> ls("package:BSgenome.Celegans.UCSC.ce2")
```

```
[1] "Celegans"
```

```
> Celegans
```

Caenorhabditis elegans genome:

```
Single sequences (DNAString objects, see '?seqnames'):  
  chrI   chrII  chrIII chrIV   chrV   chrX   chrM
```

```
Multiple sequences (BStringViews objects, see '?mseqnames'):  
  upstream1000 upstream2000 upstream5000
```

(use the '\$' or '[' operator to access a given sequence)

```
> comment(Celegans$chrI)
```

```
[1] "Caenorhabditis elegans - chromosome I (generated from FASTA file chrI.fa)"
```

Display chromosome I:

```
> Celegans$chrI
15080483-letter "DNAString" object
Value: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTA...AGGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
```

The number of letters in this sequence can be retrieved with:

```
> cI <- Celegans$chrI
> length(cI)
```

```
[1] 15080483
```

Some basic stats:

```
> af <- alphabetFrequency(cI)
> af
```

	A	C	G	T	M	R	W	S	Y	K
4838561	2697177	2693544	4851201		0	0	0	0	0	0
	V	H	D	B	N	-				
	0	0	0	0	0	0				

```
> sum(af) == length(cI)
```

```
[1] TRUE
```

## 2 Find patterns in a DNA sequence

To find all exact matches of pattern "ACCCAGGGC":

```
> p <- "ACCCAGGGC"
> countPattern(p, cI)
```

```
[1] 0
```

```
> countPattern(p, cI, mismatch = 1)
```

```
[1] 235
```

The matches can be stored in a *BStringViews* object by using the `matchPattern` method:

```
> m <- matchPattern(p, cI, mismatch = 1)
> m[4:6]
```

```
Views on a 15080483-letter DNAString subject
Subject: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...GGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
Views:
```

	start	end	width	
[1]	187350	187358	9	[ACCCAAGGC]
[2]	213236	213244	9	[ACCCAGGGG]
[3]	424133	424141	9	[ACCCAGGAC]

```
> mismatch(p, m[4:6])
```

```
[[1]]
[1] 6
```

```
[[2]]
[1] 9
```

```
[[3]]
[1] 8
```

The `mismatch` method (new in *Biostrings* 2) returns the positions of the mismatching letters.

Note: The `mismatch` method is in fact a particular case of a (vectorized) *alignment* function where only “replacements” are allowed. Current implementation is slow but this will change.

It may happen that a match is *out of limits* like here:

```
> p2 <- DNASTring("AAGCCTAAGCCTAAGCCTAA")
> m2 <- matchPattern(p2, cI, mismatch = 2)
> m2[1:4]
```

Views on a 15080483-letter DNASTring subject  
 Subject: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...GGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC  
 Views:

	start	end	width	
[1]	-1	18	20	[ GCCTAAGCCTAAGCCTAA]
[2]	5	24	20	[AAGCCTAAGCCTAAGCCTAA]
[3]	11	30	20	[AAGCCTAAGCCTAAGCCTAA]
[4]	17	36	20	[AAGCCTAAGCCTAAGCCTAA]

```
> p2 == m2[1:4]
```

```
[1] FALSE TRUE TRUE TRUE
```

```
> mismatch(p2, m2[1:4])
```

```
[[1]]
[1] 1 2
```

```
[[2]]
integer(0)
```

```
[[3]]
integer(0)
```

```
[[4]]
integer(0)
```

The list of exact matches and the list of inexact matches can both be obtained with:

```
> m2[p2 == m2]
> m2[p2 != m2]
```

Note that the length of `m2[p2 == m2]` should be equal to `countPattern(p2, cI, mismatch=0)`.

### 3 A note on performance

If needed, the `matchPattern` and `countPattern` methods convert their first argument (the pattern) to an object of the same class than their second argument (the subject) before they pass it to the function that actually implements the fast search algorithm.

So if you need to reuse the same pattern a high number of times, it's a good idea to convert it *before* to pass it to the `matchPattern` or `countPattern` method. This way the conversion is done only once:

```
> library(hgu95av2probe)
> tmpseq <- BStringViews(hgu95av2probe$sequence, "DNASTring")
> someStats <- function(v) {
+   GC <- DNASTring("GC")
+   CG <- DNASTring("CG")
+   sapply(1:length(v), function(i) {
+     y <- v[i]
+     c(alphabetFrequency(y)[1:4], GC = countPattern(GC, y),
+       CG = countPattern(CG, y))
+   })
+ }
> someStats(tmpseq[1:10])
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
A	1	5	6	4	4	2	4	5	9	2
C	10	5	4	7	5	7	10	8	7	10
G	6	5	3	8	8	6	4	5	4	4
T	8	10	12	6	8	10	7	7	5	9
GC	2	1	1	4	3	2	2	2	1	1
CG	0	0	0	2	1	1	0	0	0	0