Triplex: User Guide

Matej Lexa, Tomáš Martínek, Jiří Hon

October 13, 2014

Contents

1 Introduction

The R triplex package is essentially an R interface to the underlying C implementation of a dynamic-programming search strategy of the same name published in [1]. The main functionality of the original program is to detect positions of subsequences in a much larger sequence, where these subsequences are likely to fold into an intramolecular triplex (H-DNA). The evaluation is based on the number of canonical nucleotide triplets that can form from nucleotides in such subsequence . In creating this incarnation in R, we extended the basic functionality, to also include the calculation of likely base-pairing in the triple helices. This allowed us to extend the functionality of the package towards visualization showing the exact base-pairing in 2D or 3D as published earlier [2].

The rest of this vignette is organized as follows: The basic usage of the package for triplex detection is described in section 2. Two methods for visualization of detected triplexes in 2D and 3D are shown in section 3. Section 4 describes techniques for the conversion of search results into other types of objects such as *GRanges* or *DNAStringSet*, that can be further exported into GFF3 or FASTA files. The vignette is concluded with section 5 showing triplex package usage on a real genomic sequence from *BSGenome*.

2 Detection

As usual, before first package use, it is necessary to load the triplex library using the following command:

```
> library(triplex)
```

Identification of potential intramolecular triplex-forming sequences in DNA is performed using the *triplex.search* function. This function has one required parameter representing the studied DNA sequence in the form of a *DNAString* object and several modifying options with predefined values (see *triplex.search* help page).

Based on triplex position (forward or reverse strand) and its third strand orientation, up to 8 types of triplexes are distinguished by the *triplex.search* function. All these triplex types are depicted on figure 1. By default, the function detects all 8 types, however this behaviour can be changed by setting *type* parameter to arbitrary value or a subset of values in the range 0 to 7.

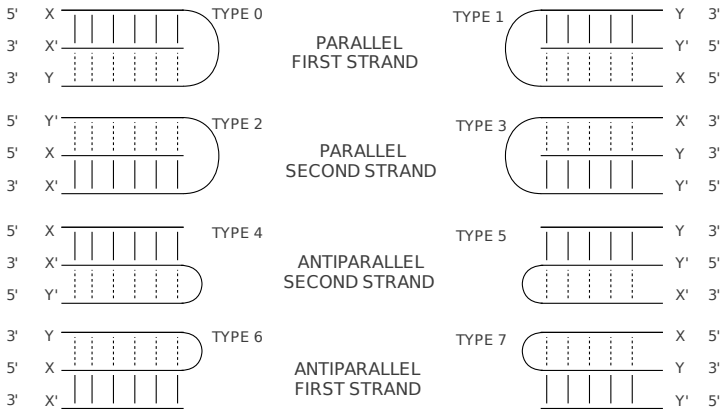


Figure 1: Triplex types

Example 1: Basic triplex detection

As a simple example, let's find all types of intramolecular triplexes in the DNA sequence *TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC*:

```
> seq <- DNAString("TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC")
> triplex.search(seq)

Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
```

```
Searching for triplex type 7...
  Triplex views on a 50-letter DNAString subject
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes:
    start width score  pvalue ins type s
[1]     3    25    15 2.9e-03   0    3 - [CCCCCTGCCTGGCATTGCTTTCCCC]
```

Detected triplexes are returned in the form of a *TriplexViews* class, which represents the basic container for storing a set of views on the same input sequence similarly to *XStringView* object (in fact *TriplexViews* only extends the *XStringView* class with the number of displayed columns). Each triplex view is defined by start locations, width, score, P-value, number of insertions, type, strand, loop start position and loop end position.

Please note, that the strand orientation depends on triplex type only. The *triplex.search* function assumes that input DNA sequence represents the forward strand.

Example 2: Selection of a specific triplex type

Let's reduce the searching procedure to triplex type 1 only, using the following command. Please note, that the output list contains potential triplexes of type 1 only.

```
> triplex.search(seq, type=1)

Searching for triplex type 1...
  Triplex views on a 50-letter DNAString subject
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes: NONE
```

The basic requirements for shape or length of detected triplexes can be defined using four parameters: *min_len*, *max_len*, *min_loop* and *max_loop*. While *min_len* and *max_len* specify the length of triplex stems composed of individual triplets, *min_loop* and *max_loop* parameters define the range of lengths for unpaired loops at the top of detected triplexes. A graphical representation of these parameters is shown in figure 2. Please note that these parameters also impacts the overall computation time. For longer triplexes, larger space has to be explored and thus more computation time is consumed.

Example 3: Definition of triplex shape

Let's modify the previous example by specifying minimal and maximal triplex lengths. Please, execute the following command and note that only one of the two triplexes detected before satisfies these conditions.

```
> triplex.search(seq, min_len=10, max_len=20)

Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
```

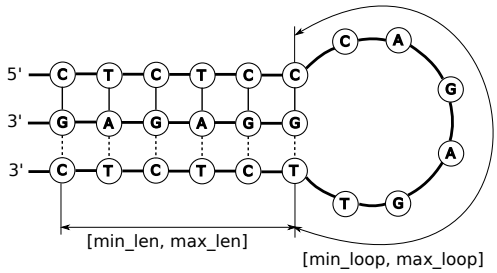


Figure 2: Triplex scheme

```
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
Searching for triplex type 7...
  Triplex views on a 50-letter DNAString subject
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes:
    start width score  pvalue ins type s
[1]     3    25    15 2.9e-03   0    3 - [CCCCCTGCCTGGCATTGCTTTCCCC]
```

The quality of each triplex is defined by its score value. A higher score value represents a higher-quality triplex. This quality is decreased by several types of imperfections at the level of triplets, such as character (nucleotide) mismatch, insertion, deletion, isomorphic group change etc. Penalization constants for these imperfections can be set up using the following parameters: *mis_pen*, *ins_pen*, *iso_pen*, *iso_bonus* and *dtwist_pen*. Detailed information about the scoring function and penalization parameters can be found in [1]. It is highly recommended to see [1] prior to changing any penalization parameter.

Example 4: Scoring function modification

Let's modify the previous example by reducing the penalization for isomorphic group change from value 5 to 2. Please execute the following command and note that calculated score values have changed.

```
> triplex.search(seq, iso_pen=2)
```

```
Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
Searching for triplex type 7...
  Triplex views on a 50-letter DNAString subject
```

```
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes:
    start width score  pvalue ins type s
[1]     3    25    16 1.3e-03   0    3 - [CCCCCTGCCTGGCATTGCTTTCCCC]
```

The *triplex.search* function can result in a large list containing tens of thousands of potential triplexes. The size of these results can be reduced using two filtration mechanisms: (1) by specifying the minimal acceptable score value using *min_score* parameter or (2) by specifying maximum acceptable P-value of results using *p_value* parameter. The P-value represents the probability of occurrence of detected triplexes in a random sequence. Calculation of P-value depends on two extreme value distribution parameters *lambda* and *mi*. It is highly recommended to see [1] prior changing the *lambda* or *mi* parameters.

Example 5: Filtration of results

Let's modify the previous example to show only triplexes with score values 14 or higher. Please execute the following command and note that only one of the two previously detected triplexes satisfies this condition.

```
> triplex.search(seq, min_score=14)

Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
Searching for triplex type 7...
  Triplex views on a 50-letter DNAString subject
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes:
    start width score  pvalue ins type s
[1]     3    25    15 2.9e-03   0    3 - [CCCCCTGCCTGGCATTGCTTTCCCC]
```

3 Visualization

Besides triplex detection, the *triplex* package offers also visualization of detected results. Three major methods of visualization are supported:

1. *Triplex alignment (text)*: Selected triplex is shown in basic text format representing the alignment of all of its strands. The output consists of four sequences: *plus* and *minus* sequences representing 5' to 3' and 3' to 5' DNA strands of the detected triplex; *anti/para-plus/minus* sequence representing the third triplex strand aligned to *plus* or *minus* strand in *antiparallel* or *parallel* fashion; and finally *loop* sequence representing the unpaired loop. Please, note that all eight triplex types shown in figure 1 can be represented using four types of alignments, because each alignment can correspond to triplex detected either on forward or reverse DNA strand.

2. *2D diagram (graphical)*: Selected triplex is shown in a 2D diagram displaying the individual triplets (based on Watson-Crick and Hoogsteen base paring) and the loop composed of unpaired nucleotides.

3. *3D model (graphical)*: Selected triplex is shown in 3D. At first, a model is calculated and then the result is displayed using the *RGL* package, which allows you to manipulate the triplex 3D model (zoom in, zoom out, rotation, etc.).

Example 6: Triplex visualization

Let's suppose, we would like to display an alignment (in text format), a 2D diagram and a 3D model of the best detected triplex from the previous examples. At first, it is suitable to store the results of calling the *triplex.search* function into an auxiliary variable.

```
> t <- triplex.search(seq)

Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
Searching for triplex type 7...

> t

  Triplex views on a 50-letter DNAString subject
subject: TTGGGGAAAGCAATGCCAGGCAGGGGGTTCCTTTCGTTACGGTCCGTCCC
triplexes:
    start width score  pvalue ins type s
[1]     3    25    15 2.9e-03   0    3 - [CCCCCTGCCTGGCATTGCTTTCCCC]
```

Then, call *triplex.alignment* function on the first item of the list. Please note that similarly to other DNA multiple sequence alignments the output of the *triplex.alignment* method is stored as *DNAStringSet* object. Also note that the *loop* sequence is always the last one and unaligned to the previous three sequences.

```
> triplex.alignment(t[1])

  A DNAStringSet instance of length 4
    width seq                                                 names
[1]    10 CCCCCTGCCT                                        minus
[2]    10 GGGGGACGGA                                        plus
[3]    10 CCCCTTTCGT                                        para-minus
[4]     5 GGCAT                                             loop
```

Then, call *triplex.diagram* function on the same item of the list. Please note that at first the triplex alignment is calculated and printed into R console and then the graphical output is displayed in a separate window. R provides methods to redirect the output to other suitable devices, such as files (see *png()*, for example).

```
> triplex.diagram(t[1])

  A DNAStringSet instance of length 4
    width seq                                                 names
[1]    10 CCCCCTGCCT                                        minus
[2]    10 GGGGGACGGA                                        plus
[3]    10 CCCCTTTCGT                                        para-minus
[4]     5 GGCAT                                             loop
```

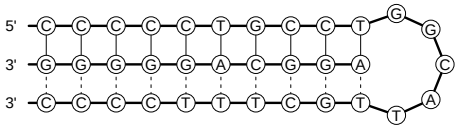


Figure 3: 2D diagram of a detected triplex

Finally, let's display the 3D structure of the same triplex using *triplex.3D* function. Please note that the result will be displayed in separate graphical window using the *RGL* library. The 3D model is based on optimizing angles and distances present in the molecule to be as close as possible to tabulated values (see [2] for more information).

```
> triplex.3D(t[1])
```

```
  A DNAStringSet instance of length 4
    width seq                                                 names
[1]    10 CCCCCTGCCT                                        minus
[2]    10 GGGGGACGGA                                        plus
[3]    10 CCCCTTTCGT                                        para-minus
[4]     5 GGCAT                                             loop
```

Figure 4: 3D scheme of detected triplex

4 Exporting Results

As mentioned above, the results of detection are stored in the *TriplexView* object. Because the *TriplexView* class is only an extension of the *XStringViews* class, all operations applied to the *XStringViews* object can also be applied to the *TriplexView* object as well.

Additionaly, *TriplexView* class contains a conversion function to create *GRanges* objects. Thus, all detected triplexes can be transformed into elements of a *GRanges* object and saved as a GFF3 file, for example.

Example 7: GRanges conversion

In this example, the output of the *triplex.search* function will be stored in a *GRanges* object and further exported as a GFF3 file. At first, let's do the conversion using the following command:

```
> gr <- as(t, "GRanges")
> gr
```

```
GRanges object with 1 range and 6 metadata columns:
      seqnames    ranges strand |     score   tritype      pvalue    lstart
         <Rle> <IRanges>  <Rle> | <integer> <integer> <character> <integer>
  [1]     chr1   [3, 27]      - |        15         3     2.9e-03        13
          lend    indels
     <integer> <integer>
  [1]        17         0
  -------
  seqinfo: 1 sequence from an unspecified genome
```

Please note that the chromosome name is set to *chr1* by default, but it can be changed to any other value. Items such as score, triplex type, P-value, loop start position, loop end position and number of indels can be added as optional attributes. In the next step the resulting *GRanges* object is exported as a GFF3 file.

```
> library(rtracklayer)
> export(gr,"test.gff", version="3")
```

Please note, that it is necessary to load the *rtracklayer* library before running the *export* command. The contents of the resulting GFF3 file are:

```
##gff-version 3
##date 2014-10-13
chr1 rtracklayer sequence_feature 3 27 15 - . tritype=3; pvalue=2.9e-03;
   lstart=13; lend=17; indels=0
```

Another possibility of utilizing the results of detection is to transform the *TriplexView* object into a *DNAStringSet* object, which represents another commonly used class of the *Biostrings* package. Triplexes stored inside *DNAStringSet* can be exported into a FASTA file, for example.

Example 8: DNAStringSet conversion

In this example, the output of the *triplex.search* function will be stored into a *DNAStringSet* object and further exported as a FASTA file. At first, let's do the conversion using the following command:

```
> dss <- as(t, "DNAStringSet")
> dss

  A DNAStringSet instance of length 1
    width seq                                                 names
[1]    25 CCCCCTGCCTGGCATTGCTTTCCCC                           start=3;end=27;sc...
```

In the next step, the *DNAStringSet* object is exported as a FASTA file.

```
> writeXStringSet(dss, file="test.fa", format="fasta")
```

The contents of the resulting FASTA file are:

```
>start=3;end=27;score=15;pvalue=2.9e-03;ins=0;type=3;strand=-
CCCCCTGCCTGGCATTGCTTTCCCC
```

Please, note that all attributes of detection such as start position, end position, score value, P-value, number of indels, triplex type and strand are stored as a *name* parameter (inside the *DNAStringSet*), and thus, they are also shown in the description line of the FASTA format (the line with the initial '>' symbol).

5 A real world example

In the following example, we load a real genomic sequence from one of the BSGenome packages, identify potential triplexes with length over 8 triplets of nucleotides and less than 15% mismatches (score >17 with the currently used scoring matrices), create three different visualizations of the best triplexes. We export the identified positions into a genome annotation track (via a GFF3 file) and FASTA file. Finally, we plot some statistics about the potential triplex distribution and nucleotide composition.

1. Load necessary libraries and genomes.

```
> library(triplex)
> library(BSgenome.Celegans.UCSC.ce10)
```

2. Search for potential triplex positions and display the results. Please note that the sequence is limited to the first 100k bases for time reasons.

```
> t <- triplex.search(Celegans[["chrX"]][1:100000],min_score=17,min_len=8)

Searching for triplex type 0...
Searching for triplex type 1...
Searching for triplex type 2...
Searching for triplex type 3...
Searching for triplex type 4...
Searching for triplex type 5...
Searching for triplex type 6...
Searching for triplex type 7...

> t

  Triplex views on a 100000-letter DNAString subject
subject: CTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTAA...AATTTTTTTTTAGGAAAAAGTTCCTTTTTTTCCT
triplexes:
     start width score  pvalue ins type s
 [1]  3164    29    21 3.6e-02   0    3 - [TTTTTTTTCAGCAAAAATTGTTTTTTTT]
 [2] 17451    29    21 3.6e-02   0    0 + [TTTTTTTTCAGCAAAAATTGTTTTTTTT]
 [3] 43641    37    28 3.9e-03   0    4 - [CAAAAAAAAAAAGAGAA...AAAAACACTTTAAAAA]
 [4] 47204    37    21 3.6e-02   0    2 - [ATTTTCCGCCGATTTTT...TTTTTGTGTTGTTTTT]
 [5] 67548    33    23 6.8e-03   0    1 + [GCTTCCTCTTCTCCTCCTCTCGCTTCTCCTCCC]
 [6] 67760    28    22 1.6e-02   0    1 + [TTCTCCTTTTCCTCATCCTCTTCCTCCT]
 [7] 76610    39    25 3.1e-02   0    4 - [AAAAAAAAAAAACGAAAA...TTATGACAAAAACAAA]
 [8] 95356    41    25 3.1e-02   0    4 - [AAAAACAAAAAAAAAAA...TTTTTAAAAAACAAAA]
 [9] 95646    36    25 3.1e-02   0    6 + [AAAAAAAAAAAAGAAAACAGAAGAAAAAATATTAAA]
[10] 95649    26    21 3.6e-02   0    3 - [TTTTTTCTTCTGTTTTCTTTTTTTTT]
[11] 97733    35    26 1.6e-02   0    5 - [AAAAAAAAAAAAATTTTTTAAATAATTTTTTTTTT]
[12] 97755    36    26 1.6e-02   0    4 - [AAAAAATATTTTTAGAAATTTTCAAAAAAAAAAAA]
[13] 97796    32    21 3.6e-02   0    1 + [TTTTTTTTTTTCAAATTTCCCTTCTTTTTTCC]
[14] 99503    45    32 2.5e-04   0    5 - [AATAAAAAAACCTTATA...AAAAAAAAAAAAAAA]
[15] 99531    44    32 2.5e-04   0    4 - [AAAAAAAAAAAAAAAAA...AATTTATAACAAACTT]
```

3. Sort the results by score and display 2D a 3D diagram of the best-scored triplex.

```
> ts <- t[order(score(t),decreasing=TRUE)]
> ts

  Triplex views on a 100000-letter DNAString subject
subject: CTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTAA...AATTTTTTTTTAGGAAAAAGTTCCTTTTTTTCCT
triplexes:
     start width score  pvalue ins type s
 [1] 99503    45    32 2.5e-04   0    5 - [AATAAAAAAACCTTATA...AAAAAAAAAAAAAAA]
 [2] 99531    44    32 2.5e-04   0    4 - [AAAAAAAAAAAAAAAAA...AATTTATAACAAACTT]
 [3] 43641    37    28 3.9e-03   0    4 - [CAAAAAAAAAAAGAGAA...AAAAACACTTTAAAAA]
 [4] 97733    35    26 1.6e-02   0    5 - [AAAAAAAAAAAAATTTTTTAAATAATTTTTTTTTT]
 [5] 97755    36    26 1.6e-02   0    4 - [AAAAAATATTTTTAGAAATTTTCAAAAAAAAAAAA]
 [6] 76610    39    25 3.1e-02   0    4 - [AAAAAAAAAAACGAAAA...TTATGACAAAAACAAA]
 [7] 95356    41    25 3.1e-02   0    4 - [AAAAACAAAAAAAAAAA...TTTTTAAAAAACAAAA]
 [8] 95646    36    25 3.1e-02   0    6 + [AAAAAAAAAAAAGAAAACAGAAGAAAAAATATTAAA]
 [9] 67548    33    23 6.8e-03   0    1 + [GCTTCCTCTTCTCCTCCTCTCGCTTCTCCTCCC]
[10] 67760    28    22 1.6e-02   0    1 + [TTCTCCTTTTCCTCATCCTCTTCCTCCT]
[11]  3164    29    21 3.6e-02   0    3 - [TTTTTTTTCAGCAAAAATTGTTTTTTTT]
[12] 17451    29    21 3.6e-02   0    0 + [TTTTTTTTCAGCAAAAATTGTTTTTTTT]
[13] 47204    37    21 3.6e-02   0    2 - [ATTTTCCGCCGATTTTT...TTTTTGTGTTGTTTTT]
[14] 95649    26    21 3.6e-02   0    3 - [TTTTTTCTTCTGTTTTCTTTTTTTT]
[15] 97796    32    21 3.6e-02   0    1 + [TTTTTTTTTTTCAAATTTCCCTTCTTTTTTCC]
```

```
> triplex.diagram(ts[1])

  A DNAStringSet instance of length 4
    width seq                                                 names
[1]    18 ATATTCCAAAAAAATAAC                                  anti-minus
[2]    18 AAAAAAAAAAAAAAAAAT                                  minus
[3]    18 TTTTTTTTTTTTTTTTTA                                  plus
[4]     9 TTTTTTTGG                                           loop
```

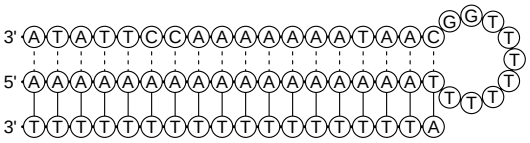


Figure 5: 2D diagram of detected triplex

```
> triplex.3D(ts[1])

  A DNAStringSet instance of length 4
    width seq                                                 names
[1]    18 ATATTCCAAAAAAATAAC                                anti-minus
[2]    18 AAAAAAAAAAAAAAAAAT                                minus
[3]    18 TTTTTTTTTTTTTTTTTA                                plus
[4]     9 TTTTTTTGG                                         loop
```

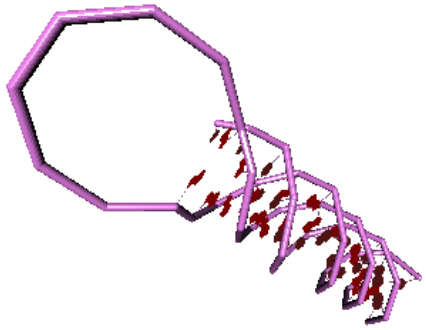


Figure 6: 3D scheme of detected triplex

4. Export all triplexes into a GFF3 format file.

```
> library(rtracklayer)
> export(as(t, "GRanges"),"test.gff", version="3")
```

The contents of the GFF3 file are as follows (the first 10 records only):

```
##gff-version 3
##date 2014-10-13
chr1 rtracklayer sequence_feature 3164 3192 21 - . tritype=3; pvalue=3.6e-02;
   lstart=3177; lend=3179; indels=0
chr1 rtracklayer sequence_feature 17451 17479 21 + . tritype=0; pvalue=3.6e-02;
   lstart=17464; lend=17466; indels=0
chr1 rtracklayer sequence_feature 43641 43677 28 - . tritype=4; pvalue=3.9e-03;
   lstart=43658; lend=43660; indels=0
chr1 rtracklayer sequence_feature 47204 47240 21 - . tritype=2; pvalue=3.6e-02;
   lstart=47221; lend=47223; indels=0
chr1 rtracklayer sequence_feature 67548 67580 23 + . tritype=1; pvalue=6.8e-03;
   lstart=67562; lend=67566; indels=0
chr1 rtracklayer sequence_feature 67760 67787 22 + . tritype=1; pvalue=1.6e-02;
   lstart=67772; lend=67775; indels=0
chr1 rtracklayer sequence_feature 76610 76648 25 - . tritype=4; pvalue=3.1e-02;
   lstart=76628; lend=76630; indels=0
chr1 rtracklayer sequence_feature 95356 95396 25 - . tritype=4; pvalue=3.1e-02;
   lstart=95374; lend=95378; indels=0
```

5. Export all triplexes into a FASTA format file.

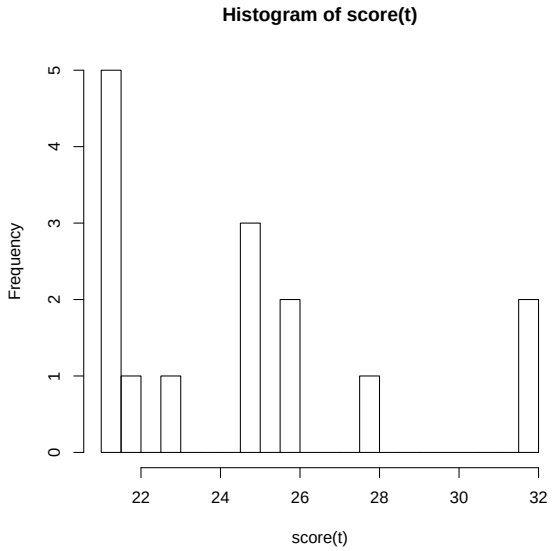
```
> writeXStringSet(as(t, "DNAStringSet"), file="test.fa", format="fasta")
```

The contents of the FASTA file are as follows (the first 10 records only):

```
>start=3164;end=3192;score=21;pvalue=3.6e-02;ins=0;type=3;strand=-
TTTTTTTTCAGCAAAAATTGTTTTTTTTT
>start=17451;end=17479;score=21;pvalue=3.6e-02;ins=0;type=0;strand=+
TTTTTTTTCAGCAAAAATTGTTTTTTTTT
>start=43641;end=43677;score=28;pvalue=3.9e-03;ins=0;type=4;strand=-
AAAAACACTTTAAAAAAAACCAAAAAAAAAAAAGAGAA
>start=47204;end=47240;score=21;pvalue=3.6e-02;ins=0;type=2;strand=-
TTTTTGTGTTGTTTTTAGCAATTTTCCGCCGATTTTT
>start=67548;end=67580;score=23;pvalue=6.8e-03;ins=0;type=1;strand=+
GCTTCCTCTTCTCCTCCTCTCGCTTCTCCTCCC
>start=67760;end=67787;score=22;pvalue=1.6e-02;ins=0;type=1;strand=+
TTCTCCTTTTCCTCATCCTCTTCCTCCT
>start=76610;end=76648;score=25;pvalue=3.1e-02;ins=0;type=4;strand=-
TTATGACAAAAACAAAAATTAAAAAAAAAAAAAACGAAAA
>start=95356;end=95396;score=25;pvalue=3.1e-02;ins=0;type=4;strand=-
TTTTTAAAAAACAAAAAAAATTTCAAAAACAAAAAAAAAAA
>start=95646;end=95681;score=25;pvalue=3.1e-02;ins=0;type=6;strand=+
AAAAAAAAAAAAGAAAACAGAAGAAAAAATATTAAA
>start=95649;end=95674;score=21;pvalue=3.6e-02;ins=0;type=3;strand=-
TTTTTTCTTCTGTTTTCTTTTTTTTT
```

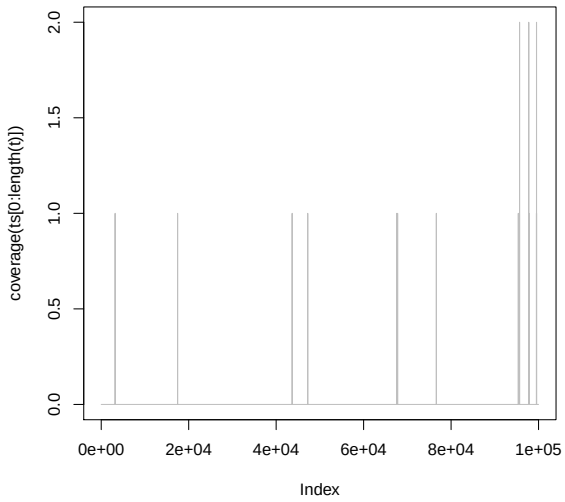
6. Show histogram for score distribution of detected triplexes.

```
> hist(score(t), breaks=20)
```



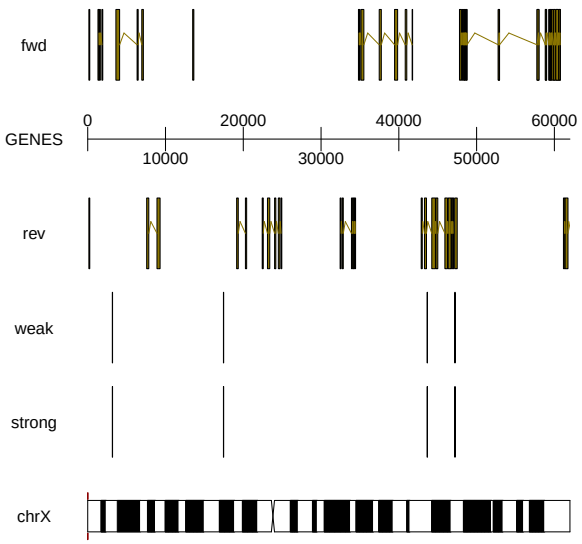
7. Show triplex distribution along the chromosome or other analysed sequence.

```
> plot(coverage(ts[0:length(t)]), type="s", col="grey75")
```



8. Show triplex distribution along the chromosome using GenomeGraphs

```
> library(GenomeGraphs)
> mart <- useMart("ensembl", dataset = "celegans_gene_ensembl")
> # Set up basic GenomeGraphs annotation tracks
> ideog <- makeIdeogram(chromosome = "X")
> genomeAxis <- makeGenomeAxis()
> genesplus <- makeGeneRegion(start = 0, end = 62000, strand = "+",
+    chromosome = "X", biomart = mart)
> genesminus <- makeGeneRegion(start = 0, end = 62000, strand = "-",
+    chromosome = "X", biomart = mart)
> # Set up triplex annotation tracks
> tall <- as(t,"GRanges")
> ta <- makeAnnotationTrack(
+   start = start(tall[which(end(tall)<62000)]),
+   end = end(tall[which(end(tall)<62000)]),
+   feature = "gene_model",
+   dp = DisplayPars(gene_model = "grey")
+ )
> ta1 <- makeAnnotationTrack(
+   start = start(tall[which(end(tall)<62000 & score(tall)>20)]),
+   end = end(tall[which(end(tall)<62000 & score(tall)>20)]),
+   feature = "gene_model",
+   dp = DisplayPars(gene_model = "darkblue")
+ )
> # Plot the entire thing
> gdPlot(list(fwd = genesplus, GENES = genomeAxis, rev = genesminus,
+    weak = ta, strong = ta1, chrX = ideog), minBase = 0,
+    maxBase = 62000, labelRot = 0)
```



6 P-value calculation

While calculating the scores of individual triplexes is straightforward with given scoring matrices and penalty scores, calculating reasonable P-values of these scores is a challenging task.

The P-values describe the probability of obtaining the reported scores by chance. To estimate it, we use a randomized genomic sequence. Because of the strikingly different nucleotide and H-DNA content of prokaryotic and eukaryotic sequences, we use E.coli genome and a segment of human chromosome 5 as models. The calculation of P-value follows the approach of [3]. We used the ExtremeValueFitHistogram() function from hmmer-2.4 to fit the values of λ and μ in the equation:

$$P\text{-}value(x) = 1 - e^{-nP(S \geq x)} \tag{1}$$

where

$$P(S \geq x) = 1 - e^{-e^{-\lambda(x-\mu)}} \tag{2}$$

The problematic part here is the determination of n. Because we search a single long sequence, but usually report multiple hits, the value of n can only be estimated. It must take into account the internal filtering of hits by *triplex* and the filtering property of the DP algorithm itself. We counted all the hits returned when fitting the EVD to a genome of size 4.8Mbp, to find an apparent value of n:

$$n(4.8Mbp) = 170000 \tag{3}$$

This leads to a reported hit every 30bp, or a ratio of n to sequence length:

$$rn = \frac{170000}{4800000} = 0.035 \tag{4}$$

References

[1] Lexa, M., Martínek, T., Burgetová, I., Kopeček, D., Brázdová, M.: *A dynamic programming algorithm for identification of triplex-forming sequences*, In: Bioinformatics, Vol. 27, No. 18, 2011, Oxford, GB, p. 2510-2517, ISSN 1367-4803.

[2] Rajdl, K. *Funkce pro manipulaci a vizualizaci molekulárních dat v prostředí R* [online].

[3] Eddy, S. R. *Maximum likelihood fitting of extreme value distributions.* 1997. Unpublished technical notes. Available at http://www.genetics.wustl.edu/eddy/publications/