Package ‘triplex’

September 18, 2025

Type Package

Title Search and visualize intramolecular triplex-forming sequences in DNA

Version 1.48.0

Date 2013-09-28

Author
Jiri Hon, Matej Lexa, Tomas Martinek and Kamil Rajdl with contributions from Daniel Kopecek

Maintainer Jiri Hon <jiri.hon@gmail.com>

Description This package provides functions for identification and visualization of potential intramolecular triplex patterns in DNA sequence. The main functionality is to detect the positions of subsequences capable of folding into an intramolecular triplex (H-DNA) in a much larger sequence. The potential H-DNA (triplexes) should be made of as many cannonical nucleotide triplets as possible. The package includes visualization showing the exact base-pairing in 1D, 2D or 3D.

License BSD_2_clause + file LICENSE

URL http://www.fi.muni.cz/~lexa/triplex/

biocViews SequenceMatching, GeneRegulation

Depends R (>= 2.15.0), S4Vectors (>= 0.5.14), IRanges (>= 2.5.27), XVector (>= 0.11.6), Biostrings (>= 2.39.10)

Imports methods, grid, GenomicRanges

Suggests rgl (>= 0.93.932), BSgenome.Celegans.UCSC.ce10, rtracklayer

LinkingTo S4Vectors, IRanges, XVector, Biostrings

git_url https://git.bioconductor.org/packages/triplex

git_branch RELEASE_3_21

git_last_commit 8434ffd

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-09-17

Contents

triplex-package . . . 2
ins . . . 3
lend . . . 4
lstart . . . 4
lwidth . . . 5
pvalue . . . 5
triplex.3D . . . 6
triplex.alignment . . . 8
triplex.diagram . . . 9
triplex.group.table . . . 10
triplex.score.table . . . 12
triplex.search . . . 13
TriplexViews-class . . . 17

Index **19**

`triplex-package` *Triplex search and visualization package*

Description

This package provides functions for the identification and visualization of potential intramolecular triplex (H-DNA) patterns in DNA sequences.

Details

This package is essentially an R interface to the underlying C implementation of a dynamic-programming search startegy of the same name (Lexa et al., 2011). The main functionality of the original program was to detect the positions of subsequences in a much larger sequence capable of folding into an intramolecular triplex (H-DNA) made of as many cannonical nucleotide triplets as possible (see `triplex.search`). In creating its incarnation in R, we extended this basic functionality, to include the calculation of exact base-pairing in the triple helices, which allowed us to extend the functionality of the package towards visualization showing the exact base-pairing in 1D, 2D or 3D (see `triplex.diagram` and `triplex.3D`).

Author(s)

Matej Lexa, Tomas Martinek, Kamil Rajdl, Jiri Hon

Maintainer: Jiri Hon <jiri.hon@google.com>

References

Lexa, M., Martinek, T., Burgetova, I., Kopecek, D., Brazdova, 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

See Also

DNAString, triplex.search, triplex.diagram, triplex.3D

Examples

```
seq <- DNAString("GGAAAGCAATGCCAGGCAGGG")
t <- triplex.search(seq, min_score=10, p_value=1)
triplex.diagram(t[1])
## Not run:
triplex.3D(t[1])

## End(Not run)
triplex.score.table()
triplex.group.table()
```

---

ins *Insertions accessor*

Description

Gets the insertion vector of an object.

Usage

```
ins(x, ...)
```

Arguments

x	An object to get the insertion values of.
...	Additional arguments.

Examples

```
t <- triplex.search(DNAString("TATTTATTTTTTCATCTTCTTTTTTTATTTTT"), max_len=11)
ins(t);
```

lend — *Loop end accessor*

Description

Gets the loop ends of an object.

Usage

```
lend(x, ...)
```

Arguments

x	An object to get the loop end values of.
...	Additional arguments.

Examples

```
t <- triplex.search(DNAString("TATTTATTTTTTCATCTTCTTTTTTTATTTTT"))
lend(t);
```

lstart — *Loop start accessor*

Description

Gets the loop starts of an object.

Usage

```
lstart(x, ...)
```

Arguments

x	An object to get the loop start values of.
...	Additional arguments.

Examples

```
t <- triplex.search(DNAString("TATTTATTTTTTCATCTTCTTTTTTTATTTTT"))
lstart(t);
```

lwidth — *Loop width accessor*

Description

Gets the loop widths of an object.

Usage

```
lwidth(x, ...)
```

Arguments

x	An object to get the loop width values of.
...	Additional arguments.

Examples

```
t <- triplex.search(DNAString("TATTTATTTTTTCATCTTCTTTTTTTATTTTT"))
lwidth(t);
```

pvalue — *P-value accessor*

Description

Gets the P-values of an object.

Usage

```
pvalue(x, ...)
```

Arguments

x	An object to get the P-values of.
...	Additional arguments.

Examples

```
t <- triplex.search(DNAString("TATTTATTTTTTCATCTTCTTTTTTTATTTTT"))
pvalue(t);
```

`triplex.3D` *Triplex visualization, 3D representation*

Description

This function visualizes a `TriplexViews` object as a 3D model. Its structure can be drawn with automatic optimalizations. To use this function, please install suggested rgl package from CRAN.

Usage

```
triplex.3D(triplex, opt = TRUE, A.col = "red", T.col = "brown",
           G.col = "green", C.col = "blue", bbone.col = "violet",
           bgr.col = "white", bbone.n = 20)
```

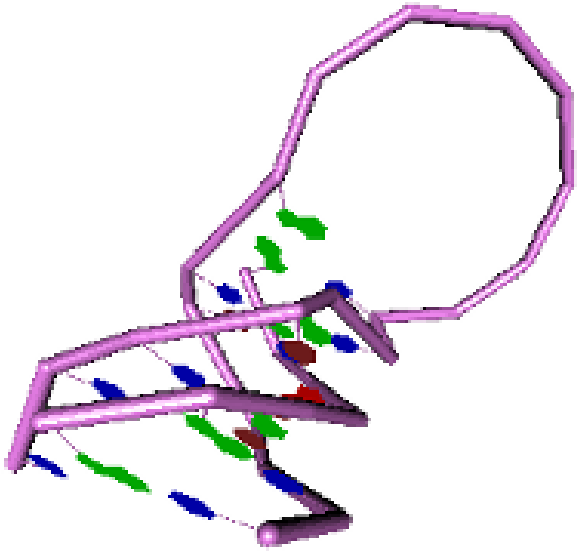
Arguments

`triplex`	`TriplexViews` object including only one triplex.
`opt`	TRUE or FALSE: TRUE - structure of triplex will be optimalized, FALSE - structure will be drawn without optimalization.
`A.col`	Color of Adine base.
`T.col`	Color of Thymin base.
`G.col`	Color of Guanine base.
`C.col`	Color of Cytosine base.
`bgr.col`	Color of background.
`bbone.col`	Color of backbone.
`bbone.n`	Number of sides of backbone bonds.

Details

The input `TriplexViews` object is required to provide additional algorithm options (see `triplex.search`). These are used for proper computation of triplex alignment.

An example of a graphical output corresponding to a triplex type 3 with DNA sequence "GGAAAG-CAATGCCAGGCAGGG" is shown in the following figure



Value

Instance of DNAStringSet object with computed alignment.

Author(s)

Kamil Rajdl, Jiri Hon

See Also

triplex.diagram, triplex.search, triplex.alignment

Examples

```
seq <- DNAString("GGAAAGCAATGCCAGGCAGGG")
t <- triplex.search(seq, min_score=10, p_value=1)
## Not run:
triplex.3D(t[1])

## End(Not run)
```

`triplex.alignment` *Triplex alignment*

Description

This function computes best triplex alignment.

Usage

```
triplex.alignment(triplex)
```

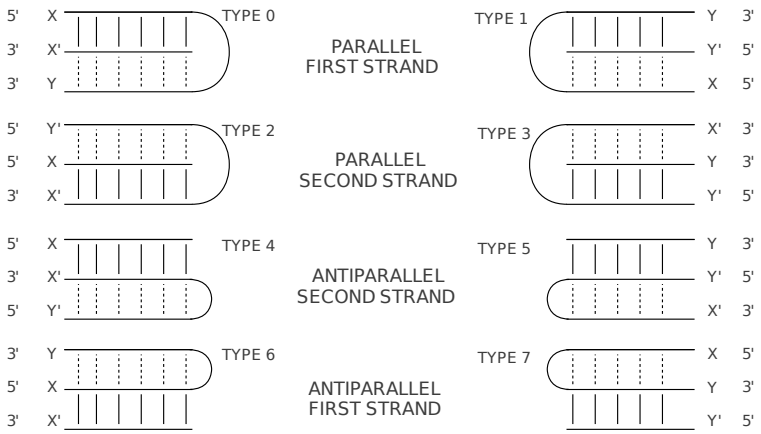
Arguments

`triplex`	`TriplexViews` object including only one triplex.

Details

Similarly to other DNA multiple sequence alignments the output of the `triplex.alignment` method is stored as `DNAStringSet` object. This object consists of four sequences: `plus` and `minus` sequences representing 5' to 3' and 3' to 5' DNA strand of detected triplex; one of the `anti-plus`, `anti-minus`, `para-plus` or `para-minus` sequence representing the third triplex strand aligned to `plus` or `minus` strand in `antiparallel` or `parallel` fashion; and finally `loop` sequence representing unpaired loop.

Please note that all eight triplex types shown in following figure can be represented using four types of alignments, because each alignment can correspond to triplex detected either on forward or reverse DNA strand.



The input `TriplexViews` object is required to provide additional algorithm options (see `triplex.search`). These are used for proper computation of triplex alignment.

Value

Instance of `DNAStringSet` object.

Author(s)

Jiri Hon

See Also

triplex.diagram, triplex.3D, triplex.search

Examples

```
seq <- DNAString("GGAAAGCAATGCCAGGCAGGG")
t <- triplex.search(seq, min_score=10, p_value=1)
triplex.alignment(t[1])
```

---

triplex.diagram *Triplex visualization, diagram representation*

Description

This function visualizes a TriplexViews object as a 2D diagram. Nucleotides are drawn as characters in circles and bonds as lines between them (Watson-Crick or Hogsteen).

Usage

```
triplex.diagram(triplex, circles = TRUE, mbonds.lty = 1,
                mbonds.lwd = 2.5, wcbonds.lty = 1, wcbonds.lwd = 1,
               hbonds.lty = 2, hbonds.lwd = 1, labels.cex = 1, circles.cex = 1,
                margin = 0.1, bonds.length = 0.07)
```

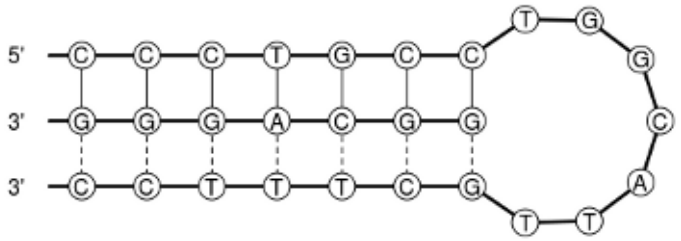
Arguments

triplex	TriplexViews object including only one triplex.
circles	TRUE or FALSE: TRUE - nucleotides are drawn as characters in circles, FALSE - nucleotides are drawn just as characters.
mbonds.lty	Type of main (skelet) bonds lines.
mbonds.lwd	Width of main (skelet) bonds lines.
wcbonds.lty	Type of Watson-Crick bonds lines.
wcbonds.lwd	Width of Watson-Crick bonds lines.
hbonds.lty	Type of Hoogsteen bonds lines.
hbonds.lwd	Width of Hoogsteen bonds lines.
labels.cex	Multiplier of size of labels of nucleotides.
circles.cex	Multiplier of size of nucleotides.
margin	Left and right margin of the picture.
bonds.length	Length of lines representing Watson-Crick and Hoogsteen bonds.

Details

The input TriplexViews object is required to provide additional algorithm options (see triplex.search). These are used for proper computation of triplex alignment.

An example of a graphical output corresponding to a triplex of type 3 with DNA sequence "GGAAAGCAATGCCAGGCAGGG" is shown in the following figure



Value

Instance of DNAStringSet object with computed alignment.

Author(s)

Kamil Rajdl, Jiri Hon

See Also

triplex.3D, triplex.search, triplex.alignment

Examples

```
seq <- DNAString("GGAAAGCAATGCCAGGCAGGG")
t <- triplex.search(seq, min_score=10, p_value=1)
triplex.diagram(t[1])
```

---

triplex.group.table *Get default isogroup tables*

Description

The triplex.group.table function returns default isogroup tables for parallel and antiparallel triplex types.

Usage

```
triplex.group.table()
```

Details

This function is used by `triplex.search` function to get default isogroup tables. These tables correspond exactly with Table 1 published in (Lexa, 2011) and they represent the isomorphic groups of triplets.

As a common triplet structure is H.WC:WC, to customize isogroup tables just use H as a row index, WC as a column index, then set desired group number and pass such modified tables through `group_table` option of `triplex.search` interface.

Value

List of two matrixes, one for parallel triplex types and the other one for antiparallel.

Note

If you modify the isogroup tables, you should consider changing also default P-value constants (`lambda`, `mu` and `rn`), because these are valid just for the default isogroup tables.

Author(s)

Tomas Martinek, Jiri Hon

References

Lexa, M., Martinek, T., Burgetova, I., Kopecek, D., Brazdova, 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

See Also

`triplex.score.table` `triplex.search`, `TriplexViews`, `triplex.diagram`, `triplex.3D`, `triplex.alignment`

Examples

```
triplex.group.table()
```

`triplex.score.table` *Get default scoring tables*

Description

The `triplex.score.table` function returns default scoring tables for parallel and antiparallel triplex types.

Usage

```
triplex.score.table()
```

Details

This function is used by `triplex.search` function to get default scoring tables. These tables correspond exactly with Table 1 published in (Lexa, 2011) and they represent the strength of Hoogsteen bonds between thirdstrand nucleotide (row index) and a nucleotide from duplex (column index).

As a common triplet structure is H.WC:WC, to customize scoring tables just use H as a row index, WC as a column index, then set desired score value and pass such modified tables through `score_table` option of `triplex.search` interface.

Please keep in mind that for a mismatch (no bond at all) special value -9 is used. If you want to change mismatch penalization, please use `mis_pen` option of `triplex.search` function.

Value

List of two matrixes, one for parallel triplex types and the other one for antiparallel.

Note

If you modify the scoring tables, you should consider changing also default P-value constants (`lambda`, `mu` and `rn`), because these are valid just for the default scoring tables.

Author(s)

Tomas Martinek, Jiri Hon

References

Lexa, M., Martinek, T., Burgetova, I., Kopecek, D., Brazdova, 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

See Also

`triplex.group.table` `triplex.search`, `TriplexViews`, `triplex.diagram`, `triplex.3D`, `triplex.alignment`

Examples

```
triplex.score.table()
```

`triplex.search` *Search intramolecular triplex-forming sequences in DNA*

Description

The `triplex.search` function identifies potential intramolecular triplex-forming sequences in DNA.

Usage

```
triplex.search(
  dna,
  type        = 0:7,
  min_score   = 15,
  p_value     = 0.05,
  min_len     = 6,
  max_len     = 25,
  min_loop    = 3,
  max_loop    = 10,
  seq_type    = 'eukaryotic',
  score_table = 'default',
  group_table = 'default',
  lambda_par  = 'default',
  lambda_apar = 'default',
  mu_par      = 'default',
  mu_apar     = 'default',
  rn_par      = 'default',
  rn_apar     = 'default',
  dtwist_pen  = 'default',
  ins_pen     = 'default',
  iso_pen     = 'default',
  iso_bonus   = 'default',
  mis_pen     = 'default')
```

Arguments

`dna`	A `DNAString` object.
`type`	Vector of triplex types (0..7) to be searched for.
`min_score`	Minimal score treshold.
`p_value`	Acceptable P-value.
`min_len`	Minimal triplex length.
`max_len`	Maximal triplex length.
`min_loop`	Minimal triplex loop length. Can not be lower than one.

`max_loop` Maximal triplex loop length.

`seq_type` Type of input sequence. Possible options: prokaryotic, eukaryotic.

`score_table` Scoring table for parallel and antiparallel triplex types. Default is the same as `triplex.score.table` output. Before changing this option, please read `triplex.score.table` help carefully.

`group_table` Isomorphic group table for parallel and antiparallel triplex types. Default is the same as `triplex.group.table` output. Before changing this option, please read `triplex.group.table` help carefully.

`lambda_par` Lambda for parallel triplex types 0,1,2,3. Default for prokaryotic sequence is 0.8892, for eukaryotic 0.8433.

`lambda_apar` Lambda for antiparallel triplex types 4,5,6,7. Default for prokaryotic sequence is 0.8092, for eukaryotic 0.6910.

`mu_par` Mu for parallel triplex types 0,1,2,3. Default for prokaryotic sequence is 7.4805, for eukaryotic 0.8433.

`mu_apar` Mu for antiparallel triplex types 4,5,6,7. Default for prokaryotic sequence is 7.6569, for eukaryotic 7.9611.

`rn_par` Hit ratio (reported hits to sequence length) for parallel triplex types 0,1,2,3.

Default for prokaryotic sequence is 0.0406, for eukaryotic 0.0304.

`rn_apar` Hit ratio (reported hits to sequence length) for antiparallel triplex types 4,5,6,7.

Default for prokaryotic sequence is 0.0273, for eukaryotic 0.0405.

`dtwist_pen` Dtwist penalization, default is 7.

`ins_pen` Insertion penalization, default is 9.

`iso_pen` Isomorphic group change penalization, default is 5.

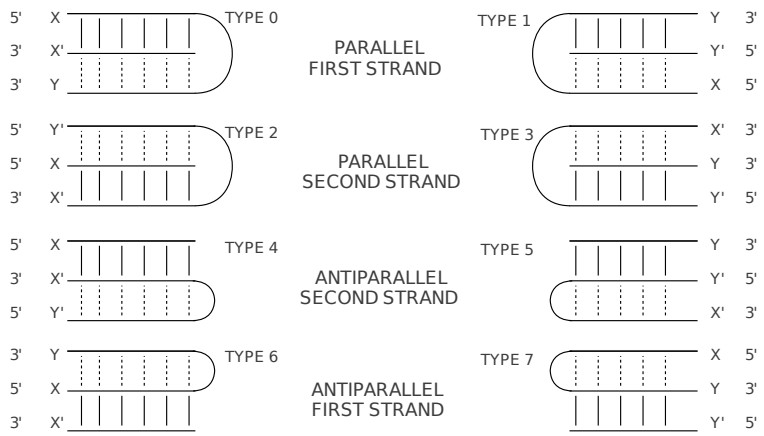
`iso_bonus` Isomorphic group stay bonus, default is 0.

`mis_pen` Mismatch penalization, default is 7.

Details

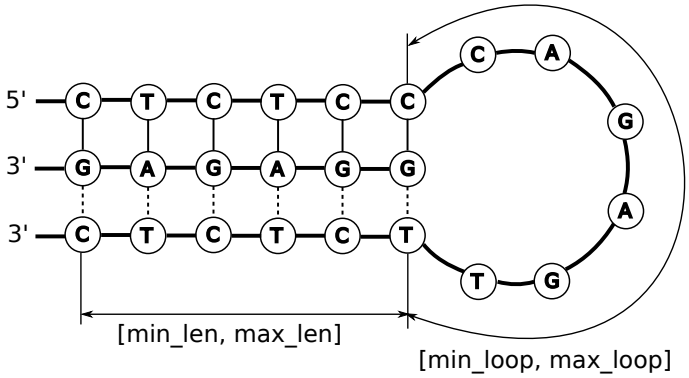
The `triplex.search` function identifies potential intramolecular triplex-forming sequences in DNA sequence represented as a `DNAString` object.

Based on triplex position (forward or reverse strand) and its third strand orientation, up to 8 types of triplexes are distinguished by the function (see the following figure). By default, the function detects all 8 types, however this behavior can be changed by setting the `type` parameter to any value or a subset of values in the range 0 to 7.



Detected triplexes are returned as instances of the `TriplexViews` class, which represents the basic container for storing a set of views on the same input sequence similarly to the `XStringViews` object (in fact `TriplexViews` only extends the `XStringViews` class with a number of displayed columns). Each triplex view is defined by start and end locations, width, score, P-value, number of insertions, type, strand, loop start and loop end. Please note, that the strand orientation depends on triplex type only. The `triplex.search` function assumes that the input DNA sequence represents the forward strand.

Basic requirements for the 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 the triplex stem composed of individual triplets, `min_loop` and `max_loop` parameters define the range of lengths for the unpaired loop at the top of the triplex. A graphical representation of these parameters is shown in the following figure. Please note, these parameters also impact the overall computation time. For longer triplexes, larger space has to be explored and thus more computation time is consumed.



The quality of each triplex is represented 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 mismatch, insertion, deletion, isomorphic group change etc. Penalization constants for these imperfections can be setup using the following parameters: `mis_pen`, `ins_pen`, `iso_pen`, `iso_bonus` and `dtwist_pen`. Detailed information about the scoring function and penal-

ization parameters can be found in (Lexa et al., 2011). It is highly recommended to see (Lexa et al., 2011) prior to changing any penalization parameters.

The `triplex.search` function can output 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 the `min_score` parameter or (2) by specifying the maximum acceptable P-value of results using the `p_value` parameter. The P-value represents the probability of occurrence of detected triplexes in random sequence. By default, only triplexes with P-value equal or less than 0.05 are reported. Calculation of P-value depends on two extreme value distribution parameters `lambda` and `mi`. By default, these parameters are set up for searching in human genome sequences. It is highly recommended to see (Lexa et al., 2011) prior to changing either of the `lambda` and `mi` parameters.

Value

Instance of `TriplexViews` object based on `XStringViews` class.

Note

If you modify the penalization options (`dtwist_pen`, `ins_pen`, `iso_pen`, `iso_bonus`, `mis_pen`), scoring tables (`score_table`) or isogroup tables (`group_table`), you should consider changing also default P-value constants (`lambda`, `mu` and `rn`) to get relevant P-values.

Author(s)

Matej Lexa, Tomas Martinek, Jiri Hon

References

Lexa, M., Martinek, T., Burgetova, I., Kopecek, D., Brazdova, 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

See Also

`TriplexViews`, `triplex.score.table` `triplex.group.table` `triplex.diagram`, `triplex.3D`, `triplex.alignment`

Examples

```
# GAA triplet repeats involved in Friedreichs's ataxia
seq <- DNAString("GAAGAAGAAGAAGAAGAAGAAGAAGAAGAA")

# Search specific triplex types (see details section)
triplex.search(seq, type=c(2,3), min_score=10, p_value=1)

# Search all triplex types
t <- triplex.search(seq, min_score=10, p_value=1)

# Sort triplexes by score
t[order(score(t), decreasing=TRUE)]
```

TriplexViews-class *The TriplexViews class*

Description

The TriplexViews class is a container for storing a set of triplexes identified in the same DNA sequence (an DNAString object). Each triplex is defined by its start/end locations, score, P-value, insertion number, type, loop start, loop end and strand identification.

TriplexViews object contains also a parameter vector plus score and group tables that stores custom algorithm options that were used for triplex search. This is necessary for proper triplex visualization by triplex.diagram and triplex.3D functions.

Details

A TriplexViews object is in fact a particular case of an XStringViews object (the TriplexViews class contains the XStringViews class) so it can be manipulated in a similar manner. See ?XStringViews for detailed information.

If you are interested in algorithm options that are stored in the TriplexViews object, see parameters of triplex.search function. These options are required by visualization functions for proper computation of triplex alignment.

Constructor

There is no public constructor for TriplexViews object as it stores search algorithm options. TriplexViews object would not be useful without algorithm options attached. For more information see description or details.

Accessor-like methods

All the accessor-like methods defined for XStringViews objects work on TriplexViews objects. In addition, the following accessors are defined for TriplexViews objects:

score(x): A vector of non-negative integers containing the scores of triplexes.

pvalue(x): A vector of non-negative doubles containing the P-values of triplexes.

ins(x): A vector of non-negative integers containing the number of insertions/deletions in triplexes.

type(x): A vector of non-negative integers containing the triplex type.

lstart(x): A vector of non-negative integers containing the triplex loop starts.

lwidth(x): A vector of non-negative integers containing the triplex loop widths.

lend(x): A vector of non-negative integers containing the triplex loop ends.

strand(x): A vector of '+' or '-' signs to identify on which strand the triplex was found.

toString(x): Converts TriplexViews object into vector of strings.

Note

The only standard way to create a TriplexViews object is to use triplex.search function.

Author(s)

Jiri Hon

See Also

triplex.search, triplex.diagram, triplex.3D, XStringViews, triplex.alignment

Examples

```
seq <- DNAString("GAAGAAGAAGAAGAAGAAGAAGAAGAAGAA")
t <- triplex.search(seq, min_score=10, p_value=1)
start(t)
end(t)
score(t)
pvalue(t)
ins(t)
type(t)

# Search triplex with maximal score
t[score(t) == max(score(t))]

# Sort triplexes by score
t[order(score(t), decreasing=TRUE)]
```

Index

∗ **classes**
TriplexViews-class, 17
∗ **interface**
triplex.search, 13
∗ **methods**
ins, 3
lend, 4
lstart, 4
lwidth, 5
pvalue, 5
TriplexViews-class, 17
∗ **package**
triplex-package, 2

as.character,TriplexViews-method
*(*TriplexViews-class*)*, 17

class:TriplexViews
*(*TriplexViews-class*)*, 17

DNAString, *3*, *13*, *14*, *17*
DNAStringSet, *7*, *8*, *10*

ins, 3
ins,TriplexViews-method
*(*TriplexViews-class*)*, 17

lend, 4
lend,TriplexViews-method
*(*TriplexViews-class*)*, 17
lstart, 4
lstart,TriplexViews-method
*(*TriplexViews-class*)*, 17
lwidth, 5
lwidth,TriplexViews-method
*(*TriplexViews-class*)*, 17

pvalue, 5
pvalue,TriplexViews-method
*(*TriplexViews-class*)*, 17

score,TriplexViews-method
*(*TriplexViews-class*)*, 17
show,TriplexViews-method
*(*TriplexViews-class*)*, 17
strand,TriplexViews-method
*(*TriplexViews-class*)*, 17

toString,TriplexViews-method
*(*TriplexViews-class*)*, 17
triplex *(*triplex-package*)*, 2
triplex-package, 2
triplex.3D, *2*, *3*, 6, *9–12*, *16–18*
triplex.alignment, *7*, 8, *10–12*, *16*, *18*
triplex.diagram, *2*, *3*, *7*, *9*, 9, *11*, *12*, *16–18*
triplex.group.table, 10, *12*, *14*, *16*
triplex.score.table, *11*, 12, *14*, *16*
triplex.search, *2*, *3*, *6–12*, 13, *17*, *18*
TriplexViews, *6*, *8–12*, *15*, *16*
TriplexViews *(*TriplexViews-class*)*, 17
TriplexViews-class, 17
TriplexViews-constructor
*(*TriplexViews-class*)*, 17
type,TriplexViews-method
*(*TriplexViews-class*)*, 17

XStringViews, *15–18*