

Package ‘SAlignR’

September 25, 2026

Version 0.10.5

Date 2026-09-25

Title Recovering Structure of Long Molecules from Structural Variation
Data

Description Implements a method to combine multiple levels of multiple
sequence alignment to uncover the structure of complex DNA rearrangements.

Depends R (>= 4.4)

Imports methods, graphics, grDevices, oompaBase, Biostrings,
NameNeedle, dendextend, ape, stringr, igraph, Polychrome,
colorspace

Suggests msa, viridisLite, R.rsp, knitr, rmarkdown

VignetteBuilder R.rsp

License Apache License (== 2.0)

URL <http://silicovore.com/OOMPA/clonefinder.html>

NeedsCompilation no

Author Kevin R. Coombes [aut, cre]

Maintainer Kevin R. Coombes <krc@silicovore.com>

Repository CRAN

Date/Publication 2026-09-25 17:00:09 UTC

Contents

AlignedCluster-class	2
Breakpoints-class	4
Cipher-class	5
Circular DNA Evidence	6
DeBruijn-class	7
SequenceCluster-class	8
StringGraph-class	10
SAlignR-data	11
Words	12

Index**14**

 AlignedCluster-class *Class "AlignedCluster"*

Description

The `AlignedCluster` class is used to align a set of clustered sequences. The `alignClusters` function creates a new object of the `AlignedCluster` class. The `alignAllClusters` function takes a `SequenceCluster` object and returns a list of `AlignedCluster` objects. Clustering is performed using the ClustalW algorithm. The associated class and functions take care of encoding and decoding sequences into a form that can be used by the implementation of ClustalW in the `msa` package.

Usage

```
alignCluster(sequences, mysub = NULL, gap0 = 10, gapE = 0.2)
alignAllClusters(sc, mysub = NULL, gap0 = 10, gapE = 0.2)
makeSubsMatrix(match = 5, mismatch = -2)
## S4 method for signature 'AlignedCluster'
image(x, counts = TRUE, col = "black", cex = 1, main = "", ...)
```

Arguments

<code>sequences</code>	A character vector that contains all sequences to be aligned.
<code>mysub</code>	A square (usually symmetric) substitution matrix.
<code>gap0</code>	A numeric value defining the penalty for opening a gap.
<code>gapE</code>	A numeric value defining the penalty for extending a gap.
<code>sc</code>	An object of the <code>SequenceCluster</code> class.
<code>match</code>	A numeric value defining the reward for matching symbols from two sequences.
<code>mismatch</code>	A numeric value defining the penalty for mismatching symbols from two sequences.
<code>x</code>	An object of the <code>AlignedCluster</code> class.
<code>col</code>	A character setting the color of annotations in the image.
<code>counts</code>	A logical value; whether to display the repeat counts of sequences on the right side of the image.
<code>main</code>	Character; the plot title.
<code>cex</code>	Numeric; size of the text inside the image of the alignment matrix.
<code>...</code>	Extra arguments for generic or plotting routines.

Value

The `alignCluster` function returns a new object of the `AlignedCluster` class. The `alignAllClusters` function returns a list of `AlignedCluster` objects. The `makeSubMatrix` function returns a symmetric substitution matrix.

Objects from the Class

Objects should be defined using the `alignCluster` or `alignAllCluster` functions. You typically pass in a character vector of sequences that have already been found to form a cluster.

Slots

alignment: A matrix of aligned sequences; rows are sequences and columns are aligned positions..

weights A numeric vector; the number of times each unique raw sequence occurs.

consensus: A character vector; the consensus sequence of a successful alignment.

Details

Alignment is performed using the implementation of the ClustalW algorithm provided by the `msa` package. The existing code to align amino-acid protein sequences is used by converting the current alphabet to one that limits its use to the known amino acids. The decision to use this method introduces a limitation: we are unable to align any set of sequences that use more than 25 distinct symbols. Attempting such an alignment will result in the `alignCluster` function returning a NULL value, which is passed on as one of the list items from `alignAllClusters`.

Note

These functions will only work if the `ms` package is installed. At the time of writing, CRAN does not install `msa` because of the way that `msa` uses the OpenMP protocol. So, `SVAlignR` only "Suggests" using the package and does not include it in the list of "Imports". Thus, to obtain this functionality, you must manually install `msa` from the yourself using the `BiocManager::install` function from `BioConductor`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Examples

```
data(longreads)
seqs <- longreads$connection[1:15]
pad <- c(rep("0", 9), rep("", 6))
names(seqs) <- paste("LR", pad, 1:length(seqs), sep = "")
seqs <- seqs[!duplicated(seqs)]
mysub <- makeSubsMatrix(match = 2, mismatch = -6)
if (!requireNamespace("msa", quietly = TRUE)) {
  warning("Cluster alignment is only available if the 'msa' package is installed.\n")
} else {
  ab <- alignCluster(seqs, mysub)
  image(ab)
}
```

Breakpoints-class	Class "Breakpoints"
-------------------	---------------------

Description

Classes for working with collections of breakpoints.

Usage

```
Breakpoints(working)
## S4 method for signature 'Breakpoints,missing'
plot(x, y, colset, ...)
```

Arguments

<code>working</code>	A data frame containing the locations of break points. These should be seven consecutive columns, starting with the break point id followed by three columns each (chromosome, start, stop) for each side of the break point.
<code>x</code>	An object of the Breakpoints class.
<code>y</code>	Anything; it is ignored.
<code>colset</code>	A character vector of color specifications.
<code>...</code>	Extra graphical parameters.

Value

The Breakpoints constructor returns a newly created object of the Breakpoints class. The plot method invisible returns its first argument.

Objects from the Class

Objects should be defined using the Breakpoints constructor. You typically pass in a data frame containing columns with the name/id of the breakpoint, and their chromosome name, start, and stop positions for each side of the break.

Slots

relLocation: A numeric vector giving relative coordinates (in the unit interval) of the breakpoints along a chromosome, with first and last break points mapped to 0 and 1.

labels: A character vector containing the names of the chromosomes.

ypos: A numeric vector indicating the chromosomes involved in the full set of break points.

spread: How far the display of different chromosomes should be spread apart on the y-axis.

id: The character vector of break point names.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

 Cipher-class

 Class "Cipher"

Description

The Cipher class is used to change between different alphabets (and so behaves as a simple substitution cipher). The Cipher function creates a new object of the Cipher class.

Usage

```
Cipher(sampleText, split = "-", extras = c("-", ":", "?" = "?"))
encode(cipher, text)
decode(cipher, text)
```

Arguments

sampleText	A character vector that contains all symbols you want to be able to transliterate. Duplicate symbols are automatically removed.
split	A single character used to split words into symbols. Defaults to a hyphen for our applications.
extras	Additional characters to be added for reverse transliteration, since they may appear as the results of alignments in consensus sequences.
cipher	An object of the Cipher class.
text	A character vector of words to be transliterated.

Value

The Cipher function returns a new object of the Cipher class. The encode and decode functions return character vectors that are the same size as their input text parameters.

Objects from the Class

Objects should be defined using the Cipher constructor. You typically pass in a character vector of "words" that contain all the symbols that are contained in the text to be translated (i.e., encoded and decoded) between languages. A standard target alphabet is created along with forward and reverse transliteration rules.

Slots

forward: A named character vector.

reverse: A named character vector.

bytes: The number of bytes used to encode each 'character' in the input test. Text with more than 72 unique characters use a two-byte encoding, which is enough for languages with up to $26 \times 72 = 1872$ characters.

Note

Attempting to manipulate a Cipher object using text containing NAs, missing values, or previously unknown symbols will result in an error.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Examples

```
motif <- "0-50-74-0-50-74-25-26-35"
alfa <- Cipher(motif)
alfa
en <- encode(alfa, motif)
en
de <- decode(alfa, en)
de
```

Circular DNA Evidence *Support and Evidence for Circular DNA*

Description

Tools to assess the evidence supporting the idea that a long-read breakpoint sequence actually represents a piece of extrachromosomal circular DNA.

Usage

```
getEvidence(cycles, seqclust, seqs, alfa, totalOnly = TRUE)
showCoverage(ID, seqs, evidence, alfa, colsams, locsams = NULL)
```

Arguments

<code>cycles</code>	A character vector containing the breakpoint strings of the potential circles to be evaluated.
<code>seqclust</code>	An object of the <code>SequenceCluster</code> class.
<code>seqs</code>	A character vector containing the breakpoint sequences of the long read sequences to be used for support of circles.
<code>alfa</code>	An object of the <code>Cipher</code> class.
<code>totalOnly</code>	A logical value; only return cycles that are totally supported?
<code>ID</code>	An integer index into the "coverage" list indicating which cycle to display.
<code>evidence</code>	A list as output by the <code>getEvidence</code> function.
<code>colsams</code>	A character vector indicating which colors to use for supporting long reads. If missing, all long reads are colored grey.
<code>locsams</code>	A character vector indicating which samples contained the long reads.

Details

TDA

Value

The `getEvidence` function returns a list with three entries:

- `coverage` is itself a list with one entry for each cycle. These entries contain `N` (an integer number of supporting reads), `RDS` (a character string with long-read identifiers separated by semicolons), `TOT` (an integer, the minimum support level for the nodes/breakpoints in the cycle), `WTOT` (an integer, the minimum weighted count of support across nodes), `COV` (an integer vector showing the number of reads supporting each individual node, and `WCOV` the weighted version of `COV`).
- `circles` is a data frame with one row for each cycle and columns `N`, `RDS`, `TOT`, `WTOT`, and `cycle`. If `totalOnly` was `TRUE`, only totally supported cycles (those with `TOT > 0`) are returned.
- `vspan` a character vector containing the strings describing an expanded set of potential cycles.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

References

TDA

See Also

TDA

Examples

```
showCoverage
getEvidence
```

DeBruijn-class

Class "DeBruijn"

Description

Classes for constructing de Bruijn graphs from collections of long read sequences mapped over breakpoints.

Usage

```
deBruijn(rawseq, M)
```

Arguments

rawseq	A character vector of the long read sequences, expressed as hyphen-separated breakpoint ids.
M	An integer; the length of the motifs/words to be used in constructing the graph.

Value

The deBruijn constructor returns a newly created object of the DeBruijn class.

Objects from the Class

Objects should be defined using the deBruijn constructor.

Slots

G: An object of the `igraph` class
adjmat: An adjacency matrix.
motifs: A table of motifs/words.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

```
SequenceCluster-class  Class "SequenceCluster"
```

Description

The SequenceCluster class is used to cluster sequences of "words" from an arbitrarily long alphabet. The SequenceCluster function returns a new object of the SequenceCluster class.

Usage

```
SequenceCluster(rawseq, method = c("needelman", "levenshtein"), NC = 5)
## S4 method for signature 'SequenceCluster,missing'
plot(x, type = "rooted", main = "Colored Clusters", ...)
updateClusters(sc, NC)
heat(x, ...)
```

Arguments

rawseq	A character vector that contains all words or "sequences" to be clustered.
method	The algorithm to use to compute distances between sequences. The choices are "levenshtein", which uses the Levenshtein edit distance, or "needelman", which uses the Needleman-Wunsch global alignment algorithm.
x	An object of the SequenceCluster class.

<code>sc</code>	An object of the SequenceCluster class.
<code>NC</code>	An integer; the number of clusters to cut from the dendrogram.
<code>type</code>	A character string; the type of plot to make. Valid types are "rooted", "clipped", or "unrooted".
<code>main</code>	Character; the plot title.
<code>...</code>	extra arguments for generic or plotting routines

Value

The SequenceCluster function returns a new object of the SequenceCluster class.

Objects from the Class

Objects should be defined using the SequenceCluster constructor. You typically pass in a character vector of "words" to be clustered.

Slots

method: A character vector describing which algorithm was used.

rawSequences A character vector that contains the input words or "sequences" that were clustered.

weights A numeric vector; the number of times each unique raw sequence occurs.

distance: A dist object.

hc: An hclust object.

NC: An integer; the number of clusters cut from the dendrogram.

clusters: An integer vector containing cluster assignments.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Examples

```
data(longreads)
sequences <- longreads$connection[1:30]      # named character vector
sequences <- sequences[!duplicated(sequences)] # dedup
sc <- SequenceCluster(sequences)             # cluster
plot(sc)                                     # visualize
sc <- updateClusters(sc, NC = 7)
plot(sc, type = "unrooted")
```

StringGraph-class *Class "StringGraph"*

Description

The StringGraph class is used to represent graphs that arise from strings representing long-read breakpoint sequences. The basic examples are: (1) "Motif Graphs" where the edges are subtring relations, and (2) "Decomposition Graphs" where the edges are restricted subtring relations that decompose a long read.

Usage

```
MotifGraph(motifNodes, alfa, name = "motif")
DecompositionGraph(decomp, alfa, motifNodes, name = "decomp")
exportSG(sg, outdir)
## S4 method for signature 'StringGraph,ANY'
plot(x, y, ...)
```

Arguments

motifNodes	A list of node names and counts, separated by length. In particular, motifNodes[[L]] should contain the nodes of length L.
alfa	A Cipher object.
name	A character vector of length one.
decomp	A decomposition object; see details.
sg	An object of the StringGraph class.
outdir	A character string, the name of the output directory.
x	An object of the StringGraph class.
y	Anything; it is ignored.
...	Extra graphical parameters.

Value

The MotifGraph and DecompositionGraph functions return a new object of the StringGraph class. The plot method and exportSG functions return nothing and are called for their side effects.

Objects from the Class

Objects should be defined using the MotifGraph or DecompositionGraph constructor. You typically pass in a "motifNodes" object, which is a list of sequence-strings separated by length, along with some auxiliary information.

Slots

name: A character vector of length one.

edgelist: A matrix representing a graph as a list of edges.

odelist: A matrix representing the nodes of the graph, along with their properties.

graph: An igraph object.

layout: A matrix containing x-y locations for the nodes.

Note

Attempting to manipulate a `StringGraph` object using text containing NAs, missing values, or previously unknown symbols will result in an error.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

SAlignR-data

SAlignR Sample Data

Description

These data sets contain binary versions of data describing breakpoints and long read sequences from an HPV-positive head-and-neck cancer sample.

Usage

```
data("longreads")
```

Format

longreads A data frame with 197 rows and 5 columns. Each row represents a single Oxford Nanopore long read from a study of a cell line from an HPV-positive head-and-neck squamous cell tumor. The five columns contain (i) a unique identifier of each long read, (ii) the length of the read, in bytes, (iii) the ordered sequence of break points, represented as a hyphen separated list of numeric identifiers, (iv) manually estimated natural groups of reads, and (v) a manually curated indication of whether certain long reads should be omitted from the analysis.

breakpoints A data frame with 82 rows and 11 columns. Each row represents a single breakpoint from a study of a cell line from an HPV-positive head-and-neck squamous cell tumor. The columns contain (1) a unique identifier that is used in the long read connections, (2-4) a description of the chromosomal segment to the left of the breakpoint, (5-7) a description of the chromosomal segment to the right of the breakpoint, (8-9) the orientation of the two chromosomal segments, (10) a shorthand description of the breakpoint with the segment names separated by a vertical bar and negative strands contained in parentheses, and (11) a shorthand representation of the reverse orientation of the breakpoint.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Source

Long read (Oxford Nanopore) sequencing was performed on samples prepared at the laboratory of Maura Gillison and David Symer. Characterization of long reads as a sequence of well-defined break points was performed by Keiko Akagi.

Examples

```
data(longreads)
head(longreads)

alphabet <- Cipher(longreads$connection)
en <- encode(alphabet, "0-50-74-0-50-74-35")
en
decode(alphabet, en)
```

Words

Class "Words"

Description

Provides the ability to find, count, and plot words of specific length in collections of strings in any sequence language.

Usage

```
makeWords(opstrings, K, nb = 1)
countWords(opstrings, K, alpha = NULL)
plotWords(K, m)
```

Arguments

opstrings	A character vector containing a set of words that have been encoded into an alphabet where each character uses the same number of bytes in the encoding.
K	An integer; the length of the words of interest.
nb	An integer; the number of bytes used to encode each character.
alpha	A Cipher object, used to decode the word-strings.
m	A list of word-counts produced by the makeWords function.

Details

For constructing motifs, or for producing De Bruijn graphs, we need to be able to decompose a set of input strings into "words" of a fixed length. In our application, the words are derived from long-read sequences that cross multiple breakpoints. Each breakpoint is given a unique name/label, that which can be of arbitrary length in order to be meaningful to the researchers. Using the `Cipher` class, we encode the breakpoint names into character strings of the same size. (In the original version of this package, we used single characters. That approach eventually proved to be inadequate when we looked at long-read data from samples with a very large number of breakpoints. We then extended the package to work with two-byte codes. This solution may eventually be extended to even longer coding sequences.)

The `makeWords` and `countWords` functions take as inputs a vector of character strings (typically describing long-read sequences) that have already been encoded into fixed-byte-length characters. They then find all words in those strings of a given fixed length. They only differ in the form of their output. The former function returns the word counts in their encoded form; the latter decodes them back to the original names (as long as you provide the optional appropriate `Cipher` argument).

The `plotWords` function gives a visible representation of words of length `K` sorted by their frequency. The x-axis contains the sorted word list; the y-axis is the frequency. The idea is that one can quickly figure out which words are most common in the input "text".

Value

The `makeWords` function returns a table of words (of length `K`) along with the counts of the number of times each one was seen in the input strings. The `countWords` function returns the same table, but with the words decoded back to the original language. The `plotWords` function returns a vector of the word counts for all words of length `K` in the list `m`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Examples

```
data(longreads)           # read sample data
raw <- longreads$connection # get the raw strings
alfa <- Cipher(raw)        # make a translation cipher
coded <- encode(alfa, raw) # encode all the input strings
makeWords(coded, 3)
countWords(coded, 3, alfa)
m <- lapply(1:8, function(J) countWords(coded, J, alfa))
plotWords(3, m)
```

Index

- * **cluster**
 - AlignedCluster-class, [2](#)
 - SequenceCluster-class, [8](#)
- * **datasets**
 - SVAlignR-data, [11](#)
- * **hplot**
 - Circular DNA Evidence, [6](#)
- * **manip**
 - Breakpoints-class, [4](#)
 - DeBruijn-class, [7](#)
 - Words, [12](#)
- * **math**
 - Cipher-class, [5](#)
 - StringGraph-class, [10](#)
- * **models**
 - Circular DNA Evidence, [6](#)
- alignAllClusters
 - (AlignedCluster-class), [2](#)
- alignCluster (AlignedCluster-class), [2](#)
- AlignedCluster-class, [2](#)
- Breakpoints (Breakpoints-class), [4](#)
- breakpoints (SVAlignR-data), [11](#)
- Breakpoints-class, [4](#)
- Cipher, [6](#), [13](#)
- Cipher (Cipher-class), [5](#)
- Cipher-class, [5](#)
- Circular DNA Evidence, [6](#)
- countWords (Words), [12](#)
- deBruijn (DeBruijn-class), [7](#)
- DeBruijn-class, [7](#)
- decode (Cipher-class), [5](#)
- DecompositionGraph (StringGraph-class), [10](#)
- encode (Cipher-class), [5](#)
- exportSG (StringGraph-class), [10](#)
- getEvidence (Circular DNA Evidence), [6](#)
- heat (SequenceCluster-class), [8](#)
- igraph, [8](#)
- image, AlignedCluster-method
 - (AlignedCluster-class), [2](#)
- longreads (SVAlignR-data), [11](#)
- makeSubsMatrix (AlignedCluster-class), [2](#)
- makeWords (Words), [12](#)
- MotifGraph (StringGraph-class), [10](#)
- plot, Breakpoints, missing-method
 - (Breakpoints-class), [4](#)
- plot, SequenceCluster, missing-method
 - (SequenceCluster-class), [8](#)
- plot, StringGraph, ANY-method
 - (StringGraph-class), [10](#)
- plotWords (Words), [12](#)
- SequenceCluster
 - (SequenceCluster-class), [8](#)
- SequenceCluster-class, [8](#)
- showCoverage (Circular DNA Evidence), [6](#)
- StringGraph (StringGraph-class), [10](#)
- StringGraph-class, [10](#)
- SVAlignR-data, [11](#)
- updateClusters (SequenceCluster-class), [8](#)
- Words, [12](#)