Package 'openPrimeR'

October 16, 2018

Title Multiplex PCR Primer Design and Analysis

Version 1.2.0

Description An implementation of methods for designing, evaluating, and comparing primer sets for multiplex PCR. Primers are designed by solving a set cover problem such that the number of covered template sequences is maximized with the smallest possible set of primers. To guarantee that high-quality primers are generated, only primers fulfilling constraints on their physicochemical properties are selected. A Shiny app providing a user interface for the functionalities of this package is provided by the 'openPrimeRui' package.

Depends R (>= 3.4.0)

License GPL-2

Encoding UTF-8

RoxygenNote 6.0.1

Imports Biostrings (>= 2.38.4), XML (>= 3.98-1.4), scales (>= 0.4.0), reshape2 (>= 1.4.1), seqinr (>= 3.3-3), IRanges (>= 2.4.8), GenomicRanges (>= 1.22.4), ggplot2 (>= 2.1.0), plyr (>= 1.8.4), dplyr (>= 0.5.0), stringdist (>= 0.9.4.1), stringr (>= 1.0.0), RColorBrewer (>= 1.1-2), DECIPHER (>= 1.16.1), lpSolveAPI (>= 5.5.2.0-17), digest (>= 0.6.9), Hmisc (>= 3.17-4), ape (>= 3.5), BiocGenerics (>= 0.16.1), S4Vectors (>= 0.8.11), foreach (>= 1.4.3), magrittr (>= 1.5), distr (>= 2.6), distrEx (>= 2.6), fitdistrplus (>= 1.0-7), uniqtag (>= 1.0), openxlsx (>= 4.0.17), grid (>= 3.1.0), grDevices (>= 3.1.0), stats (>= 3.1.0), utils (>= 3.1.0), methods (>= 3.1.0)

Suggests testthat (>= 1.0.2), knitr (>= 1.13), rmarkdown (>= 1.0), devtools (>= 1.12.0), doParallel (>= 1.0.10), pander (>= 0.6.0), learnr (>= 0.9)

SystemRequirements MAFFT (>= 7.305), OligoArrayAux (>= 3.8), ViennaRNA (>= 2.4.1), MELTING (>= 5.1.1), Pandoc (>= 1.12.3)

biocViews Software, Technology

VignetteBuilder knitr

Collate 'AnalysisStats.R' 'Comparison.R' 'Data.R' 'templates.R' 'primers.R' 'IO.R' 'IO_view.R' 'Input.R' 'Ippolito.R' 'Output.R' 'Plots.R' 'PrimerDesign.R' 'PrimerEvaluation.R'

'RefCoverage.R' 'Scoring.R' 'SettingsDoc.R' 'TemplatesDoc.R'
'Tiller.R' 'ambiguity.R' 'check_stop_codons.R'
'con_annealing_temperature.R' 'con_dimerization.R'
'con_gc_clamp.R' 'con_gc_ratio.R' 'con_melting_temperature.R'
'con_primer_coverage.R' 'con_primer_efficiency.R'
'con_primer_secondary_structures.R' 'con_repeats.R'
'con_runs.R' 'con_template_secondary_structures.R'
'constraints.R' 'constraints_eval.R' 'errors.R' 'filters.R'
'helper_functions.R' 'initialize_primers.R'
'initialize_primers_tree.R' 'openPrimeR.R' 'optimization_ILP.R'
'optimization_algo.R' 'optimization_global.R'
'optimization_greedy.R' 'plots_comparison.R' 'settings.R'
'plots_constraints.R' 'plots_coverage.R' 'plots_filtering.R'
'primer_significance.R' 'startApp.R' 'zzz.R'

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

git_branch RELEASE_3_7

git_last_commit 1341e28

git_last_commit_date 2018-04-30

Date/Publication 2018-10-15

Author Matthias Döring [aut, cre],
Nico Pfeifer [aut]

Maintainer Matthias Döring <mdoering@mpi-inf.mpg.de>

R topics documented:

`openPrimeR-package` *Multiplex PCR Primer Design and Analysis.*

Description

With openPrimeR you can evaluate existing primers or design novel primers for multiplex polymerase chain reaction that are optimized with respect to the coverage of template sequences and the physicochemical properties of the primers.

Details

For designing primers, you just need the function `design_primers` from **openPrimeR**. As a minimal input, this function requires:

A set of template sequences You an load a `Templates` object with `read_templates`.

Settings for primer design You can load a `DesignSettings` object from a (supplied) XML file with `read_settings`. The settings can be easily customized using the setters `constraints`, `constraintLimits`, `cvg_constraints`, `conOptions`, and `PCR`.

For evaluating existing primers you can load a FASTA or CSV file containing the primers and templates of of interest using `read_primers` and `read_templates`, respectively. After evaluating the properties of the primers using `check_constraints`, you can interpret the results with several functions. For example, you can analyze the coverage of the template sequences using `get_cvg_stats`, determine the deviation from the target constraints using `plot_constraint_deviation`, or create a comprehensive report on the analyzed primers using `create_report`. In order to compare several primer sets with each other, you can create a table of the properties of the primer sets using `get_comparison_table` or create a full report, again using `create_report`.

Package options

openPrimeR uses the following options:

`openPrimeR.constraint_order` The identifiers of constraints in the order they are applied during the filtering procedure. This order is maintained when loading a `DesignSettings` object.

`openPrimeR.relax_order` The identifiers of constraints in the order in which they shall be relaxed during the relaxation procedure when designing primers.

`openPrimeR.plot_abbrev` The maximal number of allowed characters for tick labels in plots.

`openPrimeR.plot_colors` A named vector providing the identifiers of RColorBrewer palettes. Each vector entry provides the plotting colors for a specific type of stratification (i.e. by run, constraint, or primer). The palettes should provide at least eight colors.

Author(s)

Maintainer: Matthias Döring <mdoering@mpi-inf.mpg.de>

Authors:

- Nico Pfeifer <pfeifer@informatik.uni-tuebingen.de>

`AnalysisStats` *Primer Analysis Statistics.*

Description

`get_cvg_ratio` Determines the ratio of template sequences that are covered by the evaluated input primers. The ratio is in the interval [0,1] where 0 indicates 0% coverage (no templates covered) and 1 indicates 100% coverage (all templates covered).

`get_cvg_stats` Retrieve statistics on covered templates, either for a single or multiple primer sets.

`get_cvg_stats_primer` Creates a table summarizing the coverage events of individual primers.

`get_comparison_table` Creates an overview of the properties of multiple primer sets by providing the inter-quartile range of primer properties in bracket notation.

Usage

```
get_cvg_ratio(primer.df, template.df, allowed.mismatches = NULL,
  cvg.definition = c("constrained", "basic"), mode.directionality = NULL,
  as.char = FALSE)

get_comparison_table(templates, primers, sample.name = NULL)

get_cvg_stats_primer(primer.df, template.df, cvg.definition = c("constrained",
  "basic"))

get_cvg_stats(primers, templates, for.viewing = FALSE,
  total.percentages = FALSE, allowed.mismatches = Inf,
  cvg.definition = c("constrained", "basic"))
```

Arguments

primer.df	A Primers object containing the primers.
template.df	A Templates object containing the template sequences corresponding to primer.df.
allowed.mismatches	The number of allowed mismatches for determining the coverage of the templates. By default, all annotated coverage events are considered.
cvg.definition	If cvg.definition is set to "constrained", the statistics for the expected coverage (after applying the coverage constraints) are retrieved. If cvg.definition is set to "basic", the coverage is determined solely by string matching (i.e. without applying the coverage constraints). By default, cvg.definition is set to "constrained".
mode.directionality	If mode.directionality is provided, the coverage of templates is computed for a specific direction of primers. Either "fw" (forward coverage only), "rev" (reverse coverage only), or "both" for both directions. By default, mode.directionality is NULL such that the directionality of the primers is determined automatically.
as.char	Whether the coverage ratio should be outputted as a percentage-formatted character vector. By default, as.char is set to FALSE such that a numeric is returned.
templates	If primers is an object of class Primers, please provide an object of class Templates containing the template sequences targeted by primers. If primers is a list, templates should be a list of Template objects.
primers	To retrieve statistics for a single primer set, please provide an object of class Primers containing a set of evaluated primers. To retrieve statistics for multiple primer sets, pelase provide a list with evaluated Primers objects.
sample.name	Either a single identifier or a character vector of identifiers for every Templates object in templates. By default, sample.name is NULL such that the Run annotations in the provided Templates objects are used.
for.viewing	Whether the table should be formatted to be human-readable. By default, for.viewing is FALSE.
total.percentages	Whether group coverage percentages should be computed in relation to the total number of template sequences or in relation to the number of templates belonging to a specific group. By default, total.percentages is FALSE suc that the percentages are group-specific.

Details

The manner in which get_cvg_ratio determines the coverage ratio depends on the directionality of the input primers. If either only forward or reverse primers are inputted, the individual coverage of each primer is used to determine the overall coverage. If, however, forward and reverse primers are inputted at the same time, the coverage is defined by the intersection of binding events from both, forward and reverse primers.

For get_cvg_stats_primer, the cells corresponding to columns with numeric identifiers indicate the percentage of coverage events occurring with a certain number of mismatches. For example column 3 provides the number of coverage events where there are exactly three mismatches between primers and templates. The column Group_Coverage provides a listing of the percentage of covered templates per group.

Value

By default, get_cvg_ratio returns a numeric providing the expected primer coverage ratio. If as.char is TRUE, the output is provided as a percentage-formatted character vector. The attributes no_covered, no_templates, and covered_templates provide the number of covered templates, the total number of templates, and the IDs of covered templates, respectively.

get_comparison_table returns a data frame summarizing the properties of the provided primer data sets.

get_cvg_stats_primer returns a list with the following entries. cvg_per_nbr_mismatches contains a data frame listing the number of binding events broken down according to the number of expected mismatches between primers and templates. cvg_per_group contains a data frame listing the the coverage of individual primers per group of templates.

get_cvg_stats returns a data frame whose entries provide the coverage of templates per group of templates.

Examples

```
data(Ippolito)
# Determine the overall coverage
cvg.ratio <- get_cvg_ratio(primer.df, template.df)
# Determine the identitity coverage ratio
cvg.ratio.0 <- get_cvg_ratio(primer.df, template.df, allowed.mismatches = 0)

# Summarize the properties of multiple primer sets
data(Comparison)
tab <- get_comparison_table(template.data[1:3], primer.data[1:3], "IGH")

data(Ippolito)
# Determine coverage stats per primer
primer.cvg.stats <- get_cvg_stats_primer(primer.df, template.df)

# Coverage statistics for a single primer set
data(Ippolito)
cvg.stats <- get_cvg_stats(primer.df, template.df)
# Coverage statistics for multiple primer sets
data(Comparison)
cvg.stats.comp <- get_cvg_stats(primer.data[1:2], template.data[1:2])
```

Data *Data Sets.*

Description

`Ippolito` IGHV primer data from Ippolito et al.

`Tiller` IGHV primer data from Tiller et al.

`Comparison` Evaluated primer sets targeting the functional human IGH immunoglobulin genes. The sets were generated using the default evaluation settings of openPrimeR. The primer sets were gathered from IMGT and the literature.

`RefCoverage` Experimental results of multiplex PCR.

Usage

```
data(Comparison)

data(Ippolito)

data(RefCoverage)

data(Tiller)
```

Format

For the `RefCoverage` data set, the `feature.matrix` data frame contains the properties of the primer set from Tiller et al. as well as a primer set that was designed by openPrimeR. The column `Experimental_Coverage` indicates the experimentally determined coverage, while the other columns relate to properties of the primers that were computed with openPrimeR. The `ref.data` list contains the raw experimental coverage of individual primers from the primer sets from Tiller and openPrimeR, which both target templates from the IGH locus. The rows of the data frames indicate primers and the columns indicate IGH templates for which experimental coverage was determined. The cell entries are hex codes. Each hex code represents a color indicating a certain experimental coverage status. Hex codes representing red shades indicate no or little amplification, while hex codes for green shades indicate high yields.

For the `Ippolito` data set, `primer.df` provides a `Primers` object containing the evaluated set of primers from Tiller et al. `template.df` provides a `Templates` object containing functional, human IGHV templates for, and `settings` provides a `DesignSettings` object providing the used analysis settings.

For the `Comparison` data set, `primer.data` and `template.data` are lists of `Primers` and `Templates` objects, respectively.

For the `Tiller` data set, `tiller.primer.df` provides a `Primers` object, `tiller.template.df` provides the corresponding `Templates` object, and `tiller.settings` provides the `DesignSettings` object that was used for evaluating `tiller.primer.df`.

References

IMGT®, the international ImMunoGeneTics information system® http://www.imgt.org (founder and director: Marie-Paule Lefranc, Montpellier, France).

Ippolito GC, Hoi KH, Reddy ST, Carroll SM, Ge X, Rogosch T, Zemlin M, Shultz LD, Ellington AD, VanDenBerg CL, Georgiou G. 2012. Antibody Repertoires in Humanized NOD-scid-IL2R gamma null Mice and Human B Cells Reveals Human-Like Diversification and Tolerance Checkpoints in the Mouse. PLoS One 7:e35497.

Tiller, Thomas, et al. "Efficient generation of monoclonal antibodies from single human B cells by single cell RT-PCR and expression vector cloning." Journal of immunological methods 329.1 (2008): 112-124.

Examples

```
# Load the comparison data
data(Comparison)
# Explore the first entry of the primer and template data:
primer.data[[1]]
template.data[[1]]
# Summarize the primer properties:
get_comparison_table(template.data, primer.data)

# Load the data from Ippolito et al.
data(Ippolito)
primer.df
template.df
constraints(settings)

# Load experimental PCR results
data(RefCoverage)

# Load the data from Tiller et al.
data(Tiller)
tiller.primer.df
tiller.template.df
constraints(tiller.settings)
```

`Input` *Input Functionalities.*

Description

`read_primers` Reads one or multiple input files with primer sequences. The input can either be in FASTA or in CSV format.

`read_templates` Read one or multiple files with template sequences in FASTA or CSV format.

`read_settings` Loads primer analysis settings from an XML file.

`Templates` The `Templates` class encapsulates a data frame containing the sequencs of the templates, their binding regions, as well as additional information (e.g. template coverage).

`Primers` The `Primers` class encapsulates a data frame representing a set of primers. Objects of this class store all properties associated with a set of primers, for example the results from evaluating the properties of a primer set or from determining its coverage.

Usage

```
Templates(...)

read_templates(fname, hdr.structure = NULL, delim = NULL,
  id.column = NULL, rm.keywords = NULL, remove.duplicates = FALSE,
  fw.region = c(1, 30), rev.region = c(1, 30), gap.char = "-",
  run = NULL)

Primers(...)

read_primers(fname, fw.id = "_fw", rev.id = "_rev",
  merge.ambig = c("none", "merge", "unmerge"), max.degen = 16,
  template.df = NULL, adapter.action = c("warn", "rm"),
  sample.name = NULL, updateProgress = NULL)

read_settings(filename = list.files(system.file("extdata", "settings", package
  = "openPrimeR"), pattern = "*.xml", full.names = TRUE), frontend = FALSE)
```

Arguments

`...`	A data frame fulfilling the structural requirements for initializing a `Templates` or `Primers` object.
`fname`	Character vector providing either a single or multiple paths to FASTA or CSV files.
`hdr.structure`	A character vector describing the information contained in the FASTA headers. In case that the headers of `fasta.file` contain template group information, please include the keyword "GROUP" in `hdr.structure`. If the numer of elements provided via `hdr.structure` is shorter than the actual header structure, the missing fields are ignored.
`delim`	Delimiter for the information in the FASTA headers.
`id.column`	Field in the header to be used as the identifier of individual template sequences.
`rm.keywords`	A vector of keywords that are used to remove templates whose headers contain any of the keywords.
`remove.duplicates`	Whether duplicate sequence shall be removed.
`fw.region`	The positional interval from the template 5' end specifying the binding sites for forward primers. The default `fw.region` is set to the first 30 bases of the templates.
`rev.region`	The positional interval from the template 3' end specifying the binding sites for reverse primers. The default `rev.region` is set to the last 30 bases of the templates.
`gap.char`	The character in the input file representing gaps. Gaps are automatically removed upon input and the default character is "-".
`run`	An identifier for the set of template sequences. By default, `run` is `NULL` and its value is set via `template.file`.
`fw.id`	For FASTA input, the identifier for forward primers in the FASTA headers.
`rev.id`	For FASTA input, the identifier for reverse primers in the FASTA headers.

merge.ambig	Indicates whether similar primers should be merged ("merge") using IUPAC ambiguity codes or whether primers should be disambiguated ("unmerge"). By default merge.ambig is set to "none", leaving primers as they are.
max.degen	A scalar numeric providing the maximum allowed degeneracy for merging primers if merge.ambig is set to "merge". Degeneracy is defined by the number of disambiguated sequences that are represented by a degenerate primer.
template.df	An object of class Templates. If template.df is provided for read_primers then the primers are checked for restriction sites upon input; otherwhise they are not checked.
adapter.action	The action to be performed when template.df is provided for identifying adapter sequences. Either "warn" to issue warning about adapter sequences or "rm" to remove identified adapter sequences. The default is "warn".
sample.name	An identifier for the input primers.
updateProgress	A Shiny progress callback function. This is NULL by default such that no progress is tracked.
filename	Path to a valid XML file containing the primer analysis settings. By default, filename is set to all settings that are shipped with openPrimeR and the lexicographically first file is loaded.
frontend	Indicates whether settings shall be loaded for the Shiny frontend. In this case no unit conversions for the PCR settings are performed. The default setting is FALSE such that the correct units are used.

Details

In the following you can find a description of the most important columns that can be found in an object of class Templates. Note that angle brackets in the column names indicate the existence of multiple possibilities.

ID The identifiers of the templates.

Identifier The internal identifiers of the templates.

Group The identifiers of the groups that the templates belong to.

Allowed_Start_<fw|rev> The start of the interval in the templates where binding is allowed for forward and reverse primers, respectively.

Allowed_End_<fw|rev> The end of the interval in the templates where binding is allowed for forward and reverse primers, respectively.

Allowed_<fw|rev> The template sequence where binding is allowed for forward and reverse primers, respectively.

Run An identifier for the set of template sequences.

Covered_By_Primers The identifiers of primers covering the templates, when the template coverage has been annotated.

primer_coverage The number of primers covering the templates, when the template coverage has been annotated.

When loading a FASTA file with read_templates, the input arguments hdr.structure, delim, id.column, rm.keywords, remove.duplicates, fw.region, rev.region, gap.character, and run are utilized. Most importantly, hdr.structure and delim should match the FASTA header structure. To learn more about setting the primer binding regions, consider the assign_binding_regions function. In contrast, when a CSV file is loaded with read_templates, the data are loaded without

performing any modifications because the CSV file should represent an object of class Templates, which can be stored using the write_templates function.

When loading primers via read_primers, the input arguments fw.id, rev.id, merge.ambig, and max.degen are only used for loading primers from a FASTA file. In this case, please ensure that fw.id and rev.id are set according to the keywords indicating the primer directionalities in the FASTA file. When loading primers from a CSV file, the format of the file should adhere to the structure defined by the Primers class.

When loading a settings file with read_settings, if filename is not provided, a default XMl settings file is loaded. Please review the function's examples to learn more about the default settings. If you want to load custom settings, you can store a modified DesignSettings object as an XML file using write_settings.

Value

The Templates constructor returns a Templates object, an instance of a data frame.

read_templates returns a single object of class Templates if a single filename was provided or a list of such objects if multiple file names were provided.

The Primers constructor returns an object of class Primers.

read_primers returns a single object of class Primers if a single input file is provided or a list of such objects if multiple files are provided.

read_settings returns an object of class DesignSettings.

Basic columns

In the following you can find a description of the most important columns that can be found in objects of class Primers. Note that angular brackets indicate the existence of multiple possibilities. The following columns are present when a set of primers is loaded from a FASTA file using read_primers:

ID The identifiers of the primers.

Identifier The internal identifiers of the primers.

Forward The sequences of forward primers.

Reverse The sequences of reverse primers.

primer_length<fw|rev> The lengths of forward and reverse primer sequences, respectively.

Direction Either 'fw' for forward primers, 'rev' for reverse primers, or 'both' for a primer pair.

Degeneracy_<fw|rev> The degeneracy (ambiguity) of forward and reverse primers, respectively.

Run An identifier describing the primer set.

Coverage-related columns

The following columns are only available in an object of class Primers after primer coverage has been computed, that is after check_constraints has been called with the active primer_coverage constraint. Computed coverage values relating solely to string matching are indicated by the prefix Basic_, while columns without this prefix relate to the coverage after applying the constraints formulated via CoverageConstraints. Information on off-target coverage events are indicated by the Off_ prefix, while on-target coverage events do not carry this prefix.

primer_coverage The number of templates that are covered by the primers. Note that if a primer set contains primers of both directions, a template is only considered covered if it is covered by primers of both directions.

`Coverage_Ratio` The ratio of templates that are covered by the primers.

`Binding_Position_Start_<fw|rev>` The upstream position in the templates where forward and reverse primers respectively bind.

`Binding_Position_End_<fw|rev>` The downstream position in the templates where forward and reverse primers respectively bind.

`Relative_<Forward|Reverse>_Binding_Position_<Start|End>_<fw|rev>` The binding upstream (`Start`) or downstream (`End`) positions of the primers relative to the forward (`Forward`) or reverse (`Reverse`) binding regions, either for forward (`fw`) or reverse primers (`rev`).

`Binding_Region_Allowed` Whether a coverage event occurred in the target binding region or not. If the allowed off-target ratio was set to 0 only coverage events within the the target region are reported.

`Nbr_of_mismatches_<fw|rev>` The number of mismatches of forward and reverse primer coverage events, respectively.

`Mismatch_pos_<fw|rev>` The position of mismatches for forward and reverse coverage events, respectively. Mismatch positions are reported relative to the 3' end, that is, position 1 indicates a mismatch in the last base of a primer.

`primer_specificity` The specificity of a primer as determined by its ratio of off-target binding events.

Constraint-related columns

Each constraint that is considered when calling `check_constraints` gives rise to at least one column in the provided `Primers` object. Due to the large number of possible constraints, we will limit our description to the `gc_clamp` constraint. Once the GC clamp property has been computed, the `gc_clamp_fw` column contains the length of the GC clamp for forward primers and `gc_clamp_rev` the corresponding length for reverse primers. Whether the desired extent of the GC clamp was obtained by a primer is indicated by the `EVAL_gc_clamp` column. It contains `TRUE` when the GC clamp constraint was fulfilled and `FALSE` when it was broken. To identify whether all required constraints were fulfilled by a primer, the `constraints_passed` column can be used. It contains `TRUE` if all `active.constraints` used by `check_constraints` were fulfilled and `FALSE` otherwise.

Examples

```
# Load a set of templates:
fasta.file <- system.file("extdata", "IMGT_data", "templates",
     "Homo_sapiens_IGH_functional_exon.fasta", package = "openPrimeR")
hdr.structure <- c("ACCESSION", "GROUP", "SPECIES", "FUNCTION")
template.df <- read_templates(fasta.file, hdr.structure, "|", "GROUP")
# Load templates from a FASTA file
fasta.file <- system.file("extdata", "IMGT_data", "templates",
          "Homo_sapiens_IGH_functional_exon.fasta", package = "openPrimeR")
hdr.structure <- c("ACCESSION", "GROUP", "SPECIES", "FUNCTION")
template.df.fasta <- read_templates(fasta.file, hdr.structure, "|", "GROUP")
# Load mutliple FASTA files
fasta.files <- c(fasta.file, fasta.file)
template.df.fastas <- read_templates(fasta.files, hdr.structure, "|", "GROUP")
# Load templates from a previously stored CSV file
csv.file <- system.file("extdata", "IMGT_data", "comparison",
               "templates", "IGH_templates.csv", package = "openPrimeR")
template.df.csv <- read_templates(csv.file)
# Load multiple CSV files:
```

```
csv.files <- c(csv.file, csv.file)
template.df.csvs <- read_templates(csv.files)
# Load a mixture of FASTA/CSV files:
mixed.files <- c(csv.file, fasta.file)
template.data <- read_templates(mixed.files)

# Load a set of primers
primer.location <- system.file("extdata", "IMGT_data", "primers", "IGHV",
                    "Ippolito2012.fasta", package = "openPrimeR")
primer.df <- read_primers(primer.location, "_fw", "_rev")

primer.fasta <- system.file("extdata", "IMGT_data", "primers", "IGHV",
                    "Ippolito2012.fasta", package = "openPrimeR")
primer.df <- read_primers(primer.fasta, "_fw", "_rev")
# Read multiple FASTA files
fasta.files <- list.files(system.file("extdata", "IMGT_data", "primers",
                "IGHV", package = "openPrimeR"), pattern = "*\\.fasta",
                full.names = TRUE)[1:3]
primer.data <- read_primers(fasta.files)
# Read primers from a CSV file
primer.csv <- system.file("extdata", "IMGT_data", "comparison",
             "primer_sets", "IGL", "IGL_openPrimeR2017.csv",  package = "openPrimeR")
primer.df <- read_primers(primer.csv)
# Read multiple primer CSV files
primer.files <- list.files(path = system.file("extdata", "IMGT_data", "comparison",
                        "primer_sets", "IGH", package = "openPrimeR"),
                         pattern = "*\\.csv", full.names = TRUE)[1:3]
primer.data <- read_primers(primer.files)
# Read a mixture of FASTA/CSV files:
mixed.primers <- c(primer.fasta, primer.csv)
primer.data <- read_primers(mixed.primers)

# Select available settings
available.settings <- list.files(
     system.file("extdata", "settings", package = "openPrimeR"),
     pattern = "*.xml", full.names = TRUE)
# Select one of the settings and load them
filename <- available.settings[1]
settings <- read_settings(filename)
```

`Output` *Output Functionalities.*

Description

`write_primers` Writes a set of primers to disk, either as a FASTA or CSV file.

`write_settings` Stores primer analysis settings to a file in XML format.

`write_templates` Stores a set of templates as a FASTA or CSV file.

`create_report` Creates a PDF report for analyzed primer sets.

`create_coverage_xls` Creation of an XLS spreadsheet providing an overview of the covered template sequences for each primer. Each cell in the spreadsheet indicates a coverage event between a primer and template using color codes. Identified coverage events are indicated by green, while primer-template pairs without coverage are indicated by red. In case that a primer

binding condition (see CoverageConstraints) was active when computing the coverage, the numeric value of the coverage condition is annotated for each cell.

Usage

```
write_templates(template.df, fname, ftype = c("FASTA", "CSV"))

write_primers(primer.df, fname, ftype = c("FASTA", "CSV"))

create_coverage_xls(primer.df, template.df, fname, settings)

create_report(primers, templates, fname, settings, sample.name = NULL,
  used.settings = NULL, ...)

write_settings(settings, fname)
```

Arguments

template.df	An object of class Templates.
fname	The path to the output file.
ftype	A character vector giving the type of the file. This can either be "FASTA" or "CSV" (default: "FASTA").
primer.df	An object of class Primers.
settings	A DesignSettings object to be stored to disk.
primers	To create a report for a single primer set, please provide an evaluated Primers object. For creating a report comparing multiple primer sets, please provide a list of Primers objects.
templates	If primers is a Primers object, templates should be a Templates object. If primers is a list of Primers objects, templates should be a list of Templates objects of the same length as primers.
sample.name	An identifier for your analysis. By default (NULL), the sample identifier is selected from the Run column of the input templates.
used.settings	A named list (with fields fw and rev) containing the relaxed settings for designing forward/reverse primers. By default (NULL), the relaxed settings are not shown in the report.
...	required.cvg (optional, default: 1), the desired coverage ratio if primers is a single primer set.

Value

write_templates stores templates to fname.

write_primers stores primers to disk.

create_coverage_xls stores information on the primer coverage in a spreadsheet.

create_report Creates a PDF file summarizing the results from analyzing one or multiple sets of primers.

write_settings returns the status from closing the connection to the output file.

Note

Creating the report requires the external programs Pandoc (http://pandoc.org) and LaTeX (http://latex-project.org).

Examples

```
data(Ippolito)
# Store templates as FASTA
fname.fasta <- tempfile("my_templates", fileext = ".fasta")
write_templates(template.df, fname.fasta)
# Store templates as CSV
fname.csv <- tempfile("my_templates", fileext = ".csv")
write_templates(template.df, fname.csv, "CSV")

data(Ippolito)
# Store primers as FASTA
fname.fasta <- tempfile("my_primers", fileext = ".fasta")
write_primers(primer.df, fname.fasta)
# Store primers as CSV
fname.csv <- tempfile("my_primers", fileext = ".csv")
write_primers(primer.df, fname.csv, "CSV")

data(Ippolito)
filename <- tempfile("cvg_overview", fileext = ".xls")
# Store coverage of a single primer in an XLS file:
my.primers <- primer.df[3,]
cvd <- unique(unlist(strsplit(my.primers$Covered_Seqs, split = ",")))
m <- match(cvd, template.df$Identifier)
my.templates <- template.df[m,]
create_coverage_xls(my.primers, my.templates, filename, settings)

setting.xml <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(setting.xml)
# Creation of a report for a single primer set
data(Ippolito)
out.file.single <- tempfile("evaluation_report", fileext = ".pdf")
create_report(primer.df, template.df, out.file.single, settings)
# Creation of a report for multiple primer sets
data(Comparison)
set.sizes <- sapply(primer.data, nrow)
sel.sets <- order(set.sizes)[1:2]
out.file.comp <- tempfile("comparison_report", fileext = ".pdf")
create_report(primer.data[sel.sets], template.data[sel.sets], out.file.comp, settings)

# Store settings to disk
xml <- settings.xml <- system.file("extdata", "settings",
       "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(xml)
out.file <- tempfile("my_settings", fileext = ".xml")
write_settings(settings, out.file)
```

`Plots` *Plotting Functions.*

Description

`plot_cvg_vs_set_size` Plots the coverage ratios of the input primer sets against the size of the sets.

`plot_penalty_vs_set_size` Plots the penalties of the input primer sets against the number of primers contained in each set. The penalties are computed using `score_primers` where more information is provided on how to set `alpha`.

`plot_primer_subsets` Visualizes the coverage of optimized primer subsets.

`plot_primer` Visualizes the binding positions of every primer relative to the target binding region in the corresponding template sequences.

`plot_template_cvg` Creates a bar plot visualizing the covered templates.

`plot_primer_cvg` Shows which groups of templates are covered by individual primers.

`plot_constraint` Shows the distribution of the primer properties. The current constraint settings are indicated with dashed lines.

`plot_constraint_fulfillment` Visualizes which primers pass the constraints and which primers break the constraints

`plot_cvg_constraints` Plots the distribution of the coverage constraint values.

`plot_constraint_deviation` Plots the deviation of primer properties from the target ranges.

`plot_primer_binding_regions` Visualizes the number of binding events of the primers with respect to the allowed binding regions in the templates.

`plot_conservation` Plots the template sequence conservation (range [0,1]) according to the Shannon entropy of the sequences.

Usage

```
plot_conservation(entropy.df, alignments, template.df, gap.char = "-")

plot_primer_binding_regions(primers, templates, direction = c("both", "fw",
  "rev"), group = NULL, relation = c("fw", "rev"),
  region.names = c("Binding region", "Amplification region"), ...)

plot_constraint(primers, settings,
  active.constraints = names(constraints(settings)), ...)

plot_constraint_fulfillment(primers, settings,
  active.constraints = names(constraints(settings)), plot.p.vals = FALSE,
  ...)

plot_cvg_constraints(primers, settings,
  active.constraints = names(cvg_constraints(settings)), ...)

plot_constraint_deviation(primers, settings,
  active.constraints = names(constraints(settings)), ...)

plot_cvg_vs_set_size(primer.data, template.data, show.labels = TRUE,
  highlight.set = NULL)

plot_penalty_vs_set_size(primer.data, settings,
  active.constraints = names(constraints(settings)), alpha = 0)

plot_primer_subsets(primer.subsets, template.df, required.cvg = 1)

plot_primer(primer.df, template.df, identifier = NULL, relation = c("fw",
  "rev"), region.names = c("Binding region", "Amplification region"))
```

```
plot_template_cvg(primers, templates, per.mismatch = FALSE, ...)

plot_primer_cvg(primers, templates, per.mismatch = FALSE, ...)
```

Arguments

entropy.df	A data frame with entropies. Each row gives the entropies of a group of related template sequences for all columns of the alignment.
alignments	A list with DNABin alignment objects corresponding to the groups (rows) in the alignment.
template.df	An object of class Templates containing the template sequences.
gap.char	The gap char in the alignments. By default, gap.char is set to "-".
primers	Either a single Primers object with evaluated primer coverage or a list containing such Primers objects.
templates	If primers is a Primers object, templates should be a Templates object. If primers is a list, then templates should be a list of Templates objects.
direction	The directionality of primers to be plotted. This can either be "both" to plot primers of both directions (the default), "fw" to plot only forward primers, or "rev" to plot only reverse primers.
group	Optional identifiers of template groups for which binding events should be determined. By default, group is set to NULL such that all templates are considered.
relation	Whether binding positions are computed relative to forward ("fw") or reverse ("rev") binding regions. The default is "fw".
region.names	Character vector of length 2 providing the names of the binding and amplification region.
...	Optional arguments groups (a character vector of groups to be plotted when primers is a single primer set), highlight.set (the identifier of a primer set to be highlighted when primers is a list), ncol (a numeric indicating the number of facet columns if primers is a list), deviation.per.primer (a boolean indicating whether constraint deviations should be plotted per primer rather than per constraint if primers is a list)
settings	An object of class DesignSettings containing the constraints to be considered.
active.constraints	A character vector containing the identifiers to be considered for plotting. By default, active.constraints is NULL such that all computed constraints found in settings are plotted.
plot.p.vals	An optional logical argument indicating whether p-values computed via primer_significance should be annotated in the plot. The default is FALSE.
primer.data	List with objects of class Primers containing the primer sets that are to be compared.
template.data	List with objects of class Templates containing the templates corresponding to primer.data.
show.labels	Whether the identifiers of the primer sets should be annotated in the plot. The default is TRUE.
highlight.set	A character vector providing the identifiers of primer sets to highlight. By default, highlight.set is NULL such that no highlighting takes place.

alpha	A numeric in the range [0,1] defining the trade-off between the maximal deviation of a constraint (large alpha) and all constraint deviations (large alpha). By default, alpha is set to 0 such that the absolute deviation across all constraints is considered.
primer.subsets	A list with optimal primer subsets, each of class Primers. The k-th list entry should correspond to an object of class Primers representing the primer subset of size k whose coverage ratio is the largest among all possible subsets of size k.
required.cvg	The required coverage ratio. The default is 100%; this value is plotted as a horizontal line.
primer.df	An object of class Primers containing primers with evaluated primer coverage.
identifier	Identifiers of primers that are to be considered. If identifier is set to NULL (the default), all primers are considered.
per.mismatch	A logical specifying whether the visualization should be stratified according to the allowed number of mismatches. By default, per.mismatch is set to FALSE such that the overall coverage is plotted.

Details

The deviations for plot_constraint_deviation are computed in the following way. Let the minimum and maximum allowed constraint values be given by the interval $[s, e]$ and the observed value be p. Then, if $p < s$, we output $-p/|s|$, if $p > e$ we output $p/|e|$, and otherwise, i.e. if $s <= p <= e$, we output 0.

The primer.subsets argument for plot_primer_subsets can be computed using subset_primer_set. The line plot indicates the ratio of covered templates when considering all primers in a primer set of a given size. The bar plots indicate the coverage ratios of individual primers in a set. The target coverage ratio is indicated by a horizontal line. Bars exceeding the target ratio possibly indicate the existence of redundant coverage events.

Value

plot_conseration returns a plot showing the degree of sequence conservation in the templates.

plot_primer_binding_regions returns a plot of the primer binding regions.

plot_constraint returns a plot showing the distribution of primer properties.

plot_constraint_fulfillment returns a plot indicating the constraints that are fulfilled by the input primers.

plot_cvg_constraints returns a plot showing the distribution of the coverage constraint values.

plot_constraint_deviation returns a plot showing the deviations of the primer properties from the target constraints.

plot_cvg_vs_set_size returns a plot of coverage vs set size.

plot_penalty_vs_set_size returns a plot of constraint penalties vs primer set sizes.

plot_primer_subsets plots the coverages of the primer subsets provided via primer.subsets.

plot_primer plots the primer binding sites in the templates.

plot_template_cvg creates a plot showing the number of covered template sequences.

plot_primer_cvg creates a plot showing the coverage of individual primers.

Note

Computing the conservation scores for using `plot_conservation` requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignment/software/).

Examples

```
data(Ippolito)
# Select binding regions for every group of templates and plot:
template.df <- select_regions_by_conservation(template.df, win.len = 30)
if (length(template.df) != 0) {
     p1 <- plot_conservation(attr(template.df, "entropies"),
                             attr(template.df, "alignments"), template.df)
}
# Select binding regions for all templates and plot:
data(Ippolito)
template.df <- select_regions_by_conservation(template.df, by.group = FALSE)
if (length(template.df) != 0) {
     p2 <- plot_conservation(attr(template.df, "entropies"),
                             attr(template.df, "alignments"), template.df)
}

# Primer binding regions of a single primer set
data(Ippolito)
p <- plot_primer_binding_regions(primer.df, template.df)
# Primer binding regions of multiple primer sets
data(Comparison)
p.comp <- plot_primer_binding_regions(primer.data[1:3], template.data[1:3])

# Plot histogram of constraints for a single primer set
data(Ippolito)
p <- plot_constraint(primer.df, settings,
                     active.constraints = c("gc_clamp", "gc_ratio"))
# Compare constraints across multiple primer sets
data(Comparison)
p.cmp <- plot_constraint(primer.data[1:3], settings,
                         active.constraints = c("gc_clamp", "gc_ratio"))

# Plot fulfillment for a single primer set:
data(Ippolito)
p <- plot_constraint_fulfillment(primer.df, settings)
# Plot fulfillment for multiple primer sets:
data(Comparison)
p.cmp <- plot_constraint_fulfillment(primer.data[1:5], settings)

# Plot coverage constraints of a single primer set
data(Ippolito)
p <- plot_cvg_constraints(primer.df, settings)
# Plot coverage constraints for mulitple primer sets
data(Comparison)
p.cmp <- plot_cvg_constraints(primer.data[1:2], settings)

# Deviations for a single primer set
data(Ippolito)
p.dev <- plot_constraint_deviation(primer.df, settings)
# Deviations for multiple primer sets
```

```
data(Comparison)
p.dev.cmp <- plot_constraint_deviation(primer.data, settings)

# Plot coverage vs primer set size
data(Comparison)
p <- plot_cvg_vs_set_size(primer.data, template.data)

# Plot penalties vs number of primers
data(Comparison)
p <- plot_penalty_vs_set_size(primer.data, settings)

# Plot the coverage of optimal primer subsets
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df, k = 3)
p <- plot_primer_subsets(primer.subsets, template.df)

# Plot of individual primer binding positions
data(Ippolito)
p <- plot_primer(primer.df[1,], template.df[1:30,])

# Visualize the template coverage of a single primer set
data(Ippolito)
p.cvg <- plot_template_cvg(primer.df, template.df)
# Stratify by allowed mismatches:
p.mm.cvg <- plot_template_cvg(primer.df, template.df, per.mismatch = TRUE)
# Compare the coverage of multiple primer sets
data(Comparison)
p.cmp.cvg <- plot_template_cvg(primer.data[1:2], template.data[1:2])
# Stratify by allowed mismatches:
p.cmp.cvg.mm <- plot_template_cvg(primer.data[1:2], template.data[1:2],
                                  per.mismatch = TRUE)

# Plot expected coverage per primer
data(Ippolito)
p.cvg <- plot_primer_cvg(primer.df, template.df)
# Plot coverage stratified by allowed mismatches:
p.cvg.mm <- plot_primer_cvg(primer.df, template.df, per.mismatch = TRUE)
# Plot coverage of multiple primer sets
data(Comparison)
p.cvg.cmp <- plot_primer_cvg(primer.data[1:3], template.data[1:3])
```

`PrimerDesign` *Primer Design Functionalities.*

Description

`design_primers` Designs a primer set maximizing the number of covered templates using the smallest possible number of primers. The algorithm tries to ensure that the designed set of primers achieves a coverage ratio not lower than `required.cvg`. To this end, the constraints for designing primers may be relaxed.

`classify_design_problem` Uses reference beta distributions of primer coverage ratios to classify a primer design task into groups ranging from *easy* to *hard*. For *easy* tasks, it should not be a problem to design a small primer set. For *hard* tasks, however, a small set of primers may not be achievable.

`get_initial_primers` Creates a set of primer candidates based on the input template sequences. This set of primers can be used to create custom primer design algorithms.

Usage

```
classify_design_problem(template.df, mode.directionality = c("both", "fw",
  "rev"), primer.length = 18, primer.estimate = FALSE, required.cvg = 1)

get_initial_primers(sample, template.df, primer.lengths,
  mode.directionality = c("fw", "rev"),
  allowed.region.definition = c("within", "any"), init.algo = c("naive",
  "tree"), max.degen = 16, conservation = 1, updateProgress = NULL)

design_primers(template.df, mode.directionality = c("both", "fw", "rev"),
  settings, init.algo = c("naive", "tree"), opti.algo = c("Greedy", "ILP"),
  required.cvg = 1, timeout = Inf, max.degen = 16, conservation = 1,
  sample.name = NULL, cur.results.loc = NULL, primer.df = NULL,
  updateProgress = NULL)
```

Arguments

`template.df`	A `Templates` object containing the template sequences with annotated primer target binding regions.
`mode.directionality`	The template strand for which primers shall be designed. Primers can be designed either for forward strands ("fw"), for reverse strands ("rev"), or for both strands ("both"). The default setting is "both".
`primer.length`	A scalar numeric providing the target length of the designed primers. The default length of generated primers is set to 18.
`primer.estimate`	Whether the number of required primers shall be estimated. By default (`FALSE`), the number of required primers is not estimated.
`required.cvg`	The desired ratio of of covered template sequences. If the target coverage ratio cannot be reached, the constraint settings are relaxed according to the the constraint limits in order to reach the target coverage. The default `required.cvg` is set to 1, indicating that 100% of the templates are to be covered.
`sample`	Character vector providing an identifier for the templates.
`primer.lengths`	Numeric vector of length 2 providing the minimal and maximal allowed lengths for generated primers.
`allowed.region.definition`	A character vector providing the definition of region where primers are to be constructed. If `allowed.region.definition` is "within", constructed primers lie within the allowed binding region. If `allowed.region.definition` is "any", primers overlap with the allowed binding region. The default is "within".
`init.algo`	The algorithm to be used for initializing primers. If `init.algo` is "naive", then primers are constructed from substrings of the input template sequences. If `init.algo` is "tree", phylogenetic trees are used to form degenerate primers whose degeneracy is bounded by `max.degen`. This option requires an installation of MAFFT (see notes). The default `init.algo` is "naive".

max.degen	The maximal degeneracy of primer candidates. This setting is particularly relevant when init.algo is set to "tree". The default setting is 16, which means that at most 4 maximally degenerate positions are allowed per primer.
conservation	Restrict the percentile of considered regions according to their conservation. Only applicable for the tree-based primer initialization. At the default of 1, all available binding regions are considered.
updateProgress	Shiny progress callback function. The default is NULL such that no progress is logged.
settings	A DesignSettings object specifying the constraint settings for designing primers.
opti.algo	The algorithm to be used for solving the primer set covering problem. If opti.algo is "Greedy" a greedy algorithm is used to solve the set cover problem. If opti.algo is "ILP" an integer linear programming formulation is used. The default opti.algo is "Greedy".
timeout	Timeout in seconds. Only applicable when opti.algo is "ILP". The default is Inf, which does not limit the runtime.
sample.name	An identifier for the primer design task. The default setting is NULL, which means that the run identifier provided in template.df is used.
cur.results.loc	Directory for storing the results of the primer design procedure. The default setting is NULL such that no output is stored.
primer.df	An optional Primers object. If an evaluated primer.df is provided, the primer design procedure only optimizes primer.df and does not perform the initialization and filtering steps. The default is NULL such that primers are initialized and filtered from scratch.

Details

classify_design_problem determines the difficulty of a primer design task by estimating the distribution of coverage ratios per primer by performing exact string matching with primers of length primer.length, which are constructed by extracting template subsequences. Next, a beta distribution is fitted to the estimated coverage distribution, which is then compare to reference distributions representing primer design problems of different difficulties via the total variance distance. The difficulty of the input primer design problem is found by selecting the class of the reference distributions that has the smallest distance to the estimated coverage distribution. An estimate of the required number of primers to reach a given required.cvg can be computed by setting primer.estimate to TRUE. Since this estimate is based solely on perfect matching primers, the number of primers that would actually be required is typically less.

The primer design algorithm used by design_primers consists of three steps: primer initialization, filtering, and optimization. The method for initializing a set of candidate primers is determined via init.algo. If init.algo is set to *naive*, primers are created by extracting substrings from all input template sequences. If init.algo is set to *tree*, degenerate primers are created by merging similar subsequences by forming their consensus sequence up to a degeneracy of at most max.degen. The tree-based initialization is recommended for related sequences.

The candidate primer set is filtered according to the constraints specified in the settings object. In some cases, it is necessary to relax the constraints in order to reach the desired required.cvg. In these cases, primers that fail the input constraints may be selected. If you would like to skip the initialization and filtering stages, you can provide an evaluated Primers object via primer.df.

Optimizing a primer set entails finding the smallest subset of primers maximizing the coverage, which is done by solving the set cover problem. If melting temperature differences are a constraint,

the optimization procedure automatically samples ranges of melting temperatures to find optimal sets for all possible temperatures. You can select the used optimization algorithm via `optia.algo`, where you can set "Greedy" for a greedy algorithm or "ILP for an integer linear program formulation (ILP). While the worst-case runtime of the greedy algorithm is shorter than the worst-case runtime of the ILP, the greedy solution may yield larger primer sets than the ILP solution.

Value

`classify_design_problem` returns a list with the following fields:

`Classification` The estimated difficulty of the primer design task.

`Class-Distances` The total variance distance of the fitted beta distribution to the reference distribution.

`Confidence` The confidence in the estimate of the design tasks' difficulty as based on the class distances.

`Uncertain` Whether the classification is highly uncertain, that is low-confidence.

`Nbr_primers_fw` **and** `Nbr_primers_rev` The respective number of required forward and reverse primers if `primer.estimate` was set to `TRUE`.

`get_initial_primers` returns a data frame with candidate primers for optimization.

`design_primers` returns a list with the following fields:

`opti`**:** A `Primers` object providing the designed primer set.

`used_constraints`**:** A list with `DesignSettings` objects for each primer direction providing the (possibly relaxed) constraints used for designing the optimal primers.

`all_results`**:** A list containing objects of class `Primers`. Each list entry corresponds to an optimal primer set for a given melting temperature.

`all_used_constraints`**:** A list containing `DesignSettings` object for each optimized set in `all_results`.

`filtered`**:** A list with data providing information on the results of the filtering procedure.

Note

Some constraints can only be computed if additional software is installed, please see the documentation of `DesignSettings` for more information. The usage of `init.algo = "tree"` requires an installation of the multiple alignment program MAFFT (http://mafft.cbrc.jp/alignment/software/).

Examples

```
# Classify the difficulty of a primer design task
data(Ippolito)
design.estimate <- classify_design_problem(template.df[1:30,])
# Estimate the number of required primers to amplify the first 5 templates
design.estimate.nbr <- classify_design_problem(template.df[1:5,], mode.directionality = "fw",
                        primer.length = 20, primer.estimate = TRUE)

data(Ippolito)
# Naive primer initialization
init.primers <- get_initial_primers("InitialPrimers", template.df,
                        c(18,18), "fw", init.algo = "naive")
# Tree-based primer initialization (requires MAFFT)
## Not run:
init.primers <- get_initial_primers("InitialPrimers", template.df,
```

```
                        c(18,18), "fw", init.algo = "tree")

## End(Not run)

# Define PCR settings and primer criteria
data(Ippolito)
# design only with minimal set of constraints
constraints(settings)$primer_length <- c("min" = 18, "max" = 18)
constraints(settings) <- constraints(settings)[c("primer_length", "primer_coverage")]
# Design only forward primers using a greedy algorithm
optimal.primers.greedy <- design_primers(template.df[1:2,], "both", settings, init.algo = "naive")
# Usage of the tree-based initialization strategy (requires MAFFT)
## Not run:
out.dir <- tempdir()
optimal.primers.tree <- design_primers(template.df[1:2,], "both", settings,
                         init.algo = "tree", opti.algo = "ILP",
                         max.degen = 16,
                         cur.results.loc = out.dir)

## End(Not run)
```

`PrimerEval` *Primer Evaluation.*

Description

`check_constraints` Determines whether a set of primers fulfills the constraints on the properties of the primers.

`check_restriction_sites` Checks a set of primers for the presence of restriction sites. To reduce the number of possible restriction sites, only unambiguous restriction sites are taken into account and only common (typically used) restriction sites are checked if a common restriction site can be found in a sequence.

`filter_primers` Filters a primer set according to the specified constraints such that all primers that do not fulfill the constraints are removed from the primer set.

`primer_significance` Uses Fisher's exact test to determine the significance of a primer set according to its ratio of fulfilled constraints.

`subset_primer_set` Determines subsets of the input primer set that are optimal with regard to the number of covered template sequences.

Usage

```
check_restriction_sites(primer.df, template.df, adapter.action = c("warn",
  "rm"), selected = NULL, only.confident.calls = TRUE,
  updateProgress = NULL)

check_constraints(primer.df, template.df, settings,
  active.constraints = names(constraints(settings)),
  to.compute.constraints = active.constraints, for.shiny = FALSE,
  updateProgress = NULL)

filter_primers(primer.df, template.df, settings,
```

```
  active.constraints = names(constraints(settings)))

subset_primer_set(primer.df, template.df, k = 1, groups = NULL,
  identifier = NULL, cur.results.loc = NULL)

primer_significance(primer.df, set.name = NULL, active.constraints = NULL)
```

Arguments

primer.df	A Primers object containing the primers whose properties are to be checked.
template.df	A Templates object containing the template sequences corresponding to primer.df.
adapter.action	The action to be performed when adapter sequences are found. Either "warn" to issue a warning about adapter sequences or "rm" to remove identified adapter sequences. Currently, only the default setting ("warn") is supported.
selected	Names of restriction sites that are to be checked. By default selected is NULL in which case all REBASE restriction sites are taken into account.
only.confident.calls	Whether only confident calls of restriction sites are returned. All restriction site call is considered *confident* if the restriction site is located in a region that does not match the template sequences. Note that this classification requires that the provided primers are somehow complementary to the provided templates. In contrast, non-confident restriction site calls are based solely on the primer sequences and do not take the templates into account, resulting in more false positive calls of restriction sites.
updateProgress	Progress callback function for shiny. The defaut is NULL meaning that no progress is monitored via the Shiny interface.
settings	A DesignSettings object containing the constraints that are to be considered.
active.constraints	A subset of the constraint identifiers provided by settings that are to be checked for fulfillment. By default active.constraints is NULL such that all constraints found in settings are evaluated. Otherwise, only the constraints specified via active.constraints that are available in settings are considered.
to.compute.constraints	Constraints that are to be computed. By default, to.compute.constraints is set to NULL such that all active.constraints are computed. If to.compute.constraints is a subset of active.constraints, all constraints specified via active.constraints are evaluated for fulfillment, but only the constraints in to.compute.constraints are newly calculated.
for.shiny	Whether the output of the function shall be formatted as HTML. The default setting is FALSE.
k	The spacing between generated primer subset sizes. By default, k is set to 1 such that all primer subsets are constructed.
groups	The identifiers of template groups according to which coverage should be determined. By default, groups is set to NULL such that all all covered templates are considered.
identifier	An identifier for storing the primer set. By default, identifier is set to NULL.
cur.results.loc	Directory for storing the results. By default, cur.results.loc is set to NULL, which means that no results are stored.
set.name	An identifier for the input primers. If NULL, the run identifier is used.

Details

When the optional argument `active.constraints` is supplied to `check_constraints`, only a subset of the constraints provided in `settings` is evaluated. Only constraints that are defined in `settings` can be computed. For a detailed description of all possible constraints and their options, please consider the `ConstraintSettings` documentation.

`subset_primer_set` determines optimal subsets of the input primer set by solving an integer-linear program. Since the quality of the primers (in terms of properties) is not taken into account when creating the subsets, this method should only be used for primer sets that are already of high quality.

`primer_significance` computes the significance by comparing the total count of fulfilled and failed constraints with the corresponding counts of primer sets from the literature. Significant p-values indicate primer sets whose rate of constraint fulfillment is higher compared to the reference sets.

Value

`check_restriction_sites` returns a data frame with possible restriction sites found in the primers.

`check_constraints` returns a `Primers` object that is augmented with columns providing the results for the evaluated constraints. The `constraints_passed` column indicates whether all `active.constraints` were fulfilled. The `EVAL_*` columns indicate the fulfillment of primer-specific constraints. The `T_EVAL_*` columns indicate the fulfillment of template-specific (e.g. coverage-based) constraints. For the coverage computations, columns prefixed by `Basic_`, indicate the results from string matching, while all other results (e.g. `primer_coverage`) indicate the expected coverage after applying the coverage constraints specified in `settings`. Columns prefixed by `Off_` indicate off-target binding results.

`filter_primers` returns a `Primers` object containing only those primers fulfilling all specified constraints.

`subset_primer_set` returns a list with optimal primer subsets, each of class `Primers`.

`primer_significance` returns a numeric providing the p-value of the primer set according to Fisher's exact test. The returned value has the following attributes:

`test` The results of the significance test

`tab` The confusion matrix for Fisher's exact test

`constraints` The names of the considered constraints

Note

Please note that some constraint computations may require the installation of additional programs; for more information please view the documentation of `DesignSettings`.

References

Roberts, R.J., Vincze, T., Posfai, J., Macelis, D. (2010) REBASE–a database for DNA restriction and modification: enzymes, genes and genomes. Nucl. Acids Res. 38: D234-D236. http://rebase.neb.com

Examples

```
data(Ippolito)
# Check the first primer for restriction sites with respect to the first 10 templates
site.df <- check_restriction_sites(primer.df[1,], template.df[1:10])
```

```
data(Ippolito)
settings.xml <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(settings.xml)
# Check GC clamp and number of runs for all primers:
constraint.df <- check_constraints(primer.df, template.df,
                    settings, active.constraints = c("gc_clamp", "no_runs"))
# Summarize the evaluation results
summary(constraint.df)

data(Ippolito)
filename <- system.file("extdata", "settings",
             "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(filename)
# Only retain the primers fulfilling the GC clamp constraint:
filtered.df <- filter_primers(primer.df, template.df, settings,
                active.constraints = c("gc_ratio"))

# Determine optimal primer subsets
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df, k = 3)

# Determine the significance of a primer set
data(Ippolito)
p.data <- primer_significance(primer.df, "Ippolito")
attr(p.data,"tab") # the confusion matrix
attr(p.data, "test") # results from Fisher's test
attr(p.data, "constraints") # considered constraints for the test
```

`runTutorial` *The openPrimeR Tutorial.*

Description

Starts a Shiny app containing the openPrimeR tutorial, which was built using the learnr package. The application starts locally and should open a new tab in your default browser. If no browser is opened, please consider the console output to identify the local port on which the server is running.

Usage

```
runTutorial(dev = FALSE)
```

Arguments

dev	A logical indicating whether to start the development version of the tutorial (default: FALSE).

Value

Opens the openPrimeR tutorial in a web browser.

Note

The Shiny app can be started only if you fulfill all of the suggested package dependencies for the Shiny framework, so please ensure that you've installed openPrimeR including all suggested dependencies.

Examples

```
# Open the tutorial
if (interactive()) {
runTutorial()
}
```

Scoring *Scoring Functions.*

Description

`score_degen` Determines the degeneration score of a sequence.

`score_conservation` Determines the sequence conservation scores of a set of templates using Shannon entropy.

`score_primers` Computes scores for a set of primers based on the deviations of the primers from the constraints.

Usage

```
score_conservation(template.df, gap.char = "-", win.len = 30,
  by.group = TRUE)

score_degen(seq, gap.char = "-")

score_primers(primer.df, settings,
  active.constraints = names(constraints(settings)), alpha = 0.5)
```

Arguments

`template.df`	A `Templates` object providing the set of templates.
`gap.char`	The gap character in the sequences. The default is "-".
`win.len`	The size of a window for evaluating conservation. The default window size is set to 30.
`by.group`	Whether the determination of binding regions should be stratified according to the groups defined in `template.df`. The default is `TRUE`.
`seq`	A list of vectors containing individual characters of a nucleotide sequence.
`primer.df`	A `Primers` object containing the primers.
`settings`	A `DesignSettings` object containing the analysis settings.
`active.constraints`	A character vector of constraint identifiers that are considered for scoring the primers.
`alpha`	A numeric that is used to determine the trade-off between the impact of the maximal observed deviation and the total deviation. At its default `alpha` is set to 0.5 such that the maximal deviation and the total deviation have an equal weight when computing the penalties.

Details

score_degen computes the degeneration of an ambiguous sequence by considering the number of unambiguous sequences that are represented by the the ambiguous sequence. Let a sequence S of length n be represented by a collection of sets such that

$$S = s_1, s_2, \ldots, s_n$$

where s_i indicates the set of unambiguous bases found at position i of the primer. Then the degeneracy D of a primer can be defined as

$$D = \prod_i |s_i|$$

where $|s_i|$ provides the number of disambiguated bases at position i.

score_primers determines the penalty of a primer in the following way. Let d be a vector indicating the absolute deviations from individual constraints and let p be the scalar penalty that is assigned to a primer. We define

$$p = \alpha \cdot \max_i d_i + \sum_i (1 - \alpha) \cdot d_i$$

such that for large values of alpha the maximal deviation dominates giving rise to a local penalty (reflecting the largest absolute deviation) and for small alpha the total deviation dominates giving rise to a global penalty (reflecting the sum of constraint deviations). When alpha is 1 only the most extreme absolute deviation is considered and when alpha is 0 the sum of all absolute deviations is computed.

Value

A list containing Entropies and Alignments. Entropies is a data frame with conservation scores. Each column indicates a position in the alignment of template sequences and each row gives the entropies of the sequences belonging to a specific group of template sequences. Alignments is a list of DNABin objects, where each object gives the alignment corresponding to one group of template sequences.

score_degen finds the number of unambiguous sequences that are represented by seq.

score_primers returns a data frame containing scores for individual primers.

Note

score_conservation requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignment/software/).

Examples

```
## Not run:
data(Ippolito)
entropy.data <- score_conservation(template.df, gap.char = "-", win.len = 18, by.group = TRUE)

## End(Not run)
# Compute degeneration for sequences with differing number of ambiguous bases
seq <- strsplit(c("ctggaattacggtacc", "taggaaccggrtaagc", "rtaaasrygtar"), split = "")
degen <- score_degen(seq)

# Score the primers
data(Ippolito)
primer.scores <- score_primers(primer.df, settings)
```

Settings *Settings Functionalities.*

Description

`DesignSettings` The `DesignSettings` class encapsulates all settings for designing and evaluating primer sets. Upon loading an XML file, the `DesignSettings` class checks whether the defined constraints can be applied by identifying whether the requirements for external programs are fulfilled. If the requirements are not fulfilled, the affected constraints are removed from the loaded `DesignSettings` object and a warning is issued. The loaded constraints are automatically ordered according to the option `openPrimeR.constraint_order` such that the runtime of the `design_primers` and `filter_primers` functions is optimized.

`constraints` Gets the active constraints of the provided `DesignSettings` object.

`constraints<-` Sets the active constraints of the provided `DesignSettings` object.

`cvg_constraints` Gets the coverage constraints of the provided `DesignSettings` object.

`cvg_constraints<-` Sets the coverage constraints of the provided `DesignSettings` object.

`conOptions` Gets the constraint settings of the provided `DesignSettings` object.

`conOptions<-` Sets the constraint settings of the provided `DesignSettings` object.

`constraintLimits` Gets the constraint limits that are defined in the provided `DesignSettings` object.

`constraintLimits<-` Sets the constraint limits of the provided `DesignSettings` object.

`PCR` Gets the PCR conditions that are defined in the provided `DesignSettings` object.

`PCR<-` Sets the PCR conditions that are defined in the provided `DesignSettings` object.

`ConstraintSettings` The `ConstraintSettings` class encapsulates the constraints on the physicochemical properties of primers.

`CoverageConstraints` The `CoverageConstraints` class encapsulates the conditions under which the coverage of primers is evaluated.

`PCR_Conditions` The `PCR_Conditions` class encapsulates the PCR conditions for the computation of primer properties.

`ConstraintOptions` The `ConstraintOptions` class encapsulates the options for constraint computations.

`parallel_setup` Registers the specified number of cores with the parallel backend.

Usage

```
constraints(x)

## S4 method for signature 'DesignSettings'
constraints(x)

## S4 method for signature 'AbstractConstraintSettings'
constraints(x)

cvg_constraints(x)

## S4 method for signature 'DesignSettings'
```

```
cvg_constraints(x)

PCR(x)

## S4 method for signature 'DesignSettings'
PCR(x)

conOptions(x)

## S4 method for signature 'DesignSettings'
conOptions(x)

constraintLimits(x)

## S4 method for signature 'DesignSettings'
constraintLimits(x)

constraints(x) <- value

## S4 replacement method for signature 'DesignSettings,list'
constraints(x) <- value

## S4 replacement method for signature 'AbstractConstraintSettings,list'
constraints(x) <- value

cvg_constraints(x) <- value

## S4 replacement method for signature 'DesignSettings'
cvg_constraints(x) <- value

constraintLimits(x) <- value

## S4 replacement method for signature 'DesignSettings'
constraintLimits(x) <- value

PCR(x) <- value

## S4 replacement method for signature 'DesignSettings'
PCR(x) <- value

conOptions(x) <- value

## S4 replacement method for signature 'DesignSettings'
conOptions(x) <- value

parallel_setup(cores = NULL)
```

Arguments

x	A DesignSettings object.
value	An object to be used in one of the setters. For constraints<- and constraintLimits<-, a list with constraint settings or boundaries. Each list entry should have a per-

missible name and consist of at most two values providing the minimal and/or maximal allowed values, which have to be denominated via `min` and `max`.

For `conOptions<-`, a list with constraint options. The permissible fields of the list and their types are documented in the `ConstraintOptions` class.

For `cvg_constraints<-`, a list with coverage constraints. Each list entry must have a permissible name and contain a numeric vector with at most two components describing the minimal and/or maximal required values that are to be indicated via `min` and `max`. The permissible contraint identifiers are documented in the `CoverageConstraints` class.

For `PCR<-`, a named list providing PCR conditions The permissible fields of the list and their types are documented in the `PCR_Conditions` class.

`cores` A numeric providing the number of cores to use. The default is `NULL` such that half the number of available cores are used.

Details

Note that for the `DesignSettings` class, the fields `Input_Constraints`, `Input_Constraint_Boundaries`, and `Coverage_Constraints` should contain entries with at most two components using the fields `min` and/or `max`. The `Input_Constraint_Boundaries` should always be at least as general as the specified `Input_Constraints`.

For an overview of permissible constraints, please consider the `ConstraintSettings` documentation.

Value

The `ConstraintSettings` constructor defines a new `ConstraintSettings` object.

The `CoverageConstraints` constructor initializes a new `CoverageConstraints` object.

The `ConstraintOptions` constructor returns a new `ConstraintOptions` object.

The `PCR_Conditions` constructor defines a new `PCR_Conditions` object.

The `DesignSettings` constructor defines a `DesignSettings` object.

`constraints` gets a list with the active constraint settings.

`cvg_constraints` returns the list of active coverage constraints.

`PCR` gets the list of PCR conditions defined in the provided `DesignSettings` object.

`conOptions` returns a list with constraint options.

`constraintLimits` gets the list of constraint limits.

`constraints<-` sets the list of constraints in a `DesignSettings` object.

`cvg_constraints<-` sets the list of coverage constraints in the provided `DesignSettings` object.

`constraintLimits<-` sets the list of constraint limits in the provided `DesignSettings` object.

`PCR<-` sets the constraint options in the provided `DesignSettings` object.

`conOptions<-` sets the specified list of constraint options in the provided `DesignSettings` object.

`parallel_setup` returns `NULL`.

Slots

`Input_Constraints` A `ConstraintSettings` object specifying the desired target value ranges for primer properties.

`Input_Constraint_Boundaries` A `ConstraintSettings` object specifying the limits for relaxing the constraints during the primer design procedure. This slot may contain the same fields as the `Input_Constraints` slot, but the specified desired ranges should be at least as general as those specified in the `Input_Constraints` slot.

`Coverage_Constraints` A `CoverageConstraints` object specifying the constraints for computing the primer coverage.

`PCR_conditions` A `PCR_Conditions` object specifying the PCR-related settings.

`constraint_settings` A `ConstraintSettings` object providing options for the computation of individual physicochemical properties.

`status` Named boolean vector indicating which of the possible constraints are active (`TRUE`) and which are not (`FALSE`).

`settings` For `ConstraintSettings`, a named list containing the settings for the active constraints. The list may contain the following fields:

primer_coverage: The required number of covered template sequences per primer.

primer_specificity: The required required specificity of primers in terms of a ratio in the interval [0,1].

primer_length: The required lengths of primer sequences.

gc_clamp: The desired number of GCs at primer 3' termini.

gc_ratio: The desired ratio of GCs in primers in terms of numbers in the interval [0,1].

no_runs: The accepted length homopolymer runs in a primer.

no_repeats: The accepted length of dinucleotide repeats in a primer.

self_dimerization: The lowest acceptable free energy [kcal/mol] for the interaction of a primer with itself. The identification of self dimers requires the software *OligoArrayAux* (see notes).

melting_temp_range: The desired melting temperature (Celsius) of primers. The accurate computation of melting temperatures requires the software *MELTING* (see notes).

melting_temp_diff: The maximal allowed difference between the melting temperatures (Celsius) of primers contained in the same set. The accurate computation of melting temperatures requires the software *MELTING* (see notes).

cross_dimerization: The lowest acceptable free energy [kcal/mol] for the interaction of a primer with another primer. The identification of cross dimers requires the software *OligoArrayAux* (see notes).

secondary_structure: The lowest acceptable free energy [kcal/mol] for the formation of primer secondary structures. Secondary structures are determined using the software *ViennaRNA* (see notes).

For `PCR_Conditions`, a named list with PCR conditions. The following fields are possible:

`use_taq_polymerase`**:** A logical identifying whether you are performing PCR with a Taq polymerase (`TRUE`) or not (`FALSE`).

`annealing_temp`**:** The annealing temperature in Celsius that is to be used for evaluating the constraints defined in the `ConstraintSettings` object. If the annealing temperature field is not provided, a suitable annealing temperature is automatically computed using a rule of thumb (i.e. subtracting 5 from the melting temperature).

`Na_concentration`**:** The molar concentration of monovalent sodium ions.

`Mg_concentration`**:** The molar concentration of divalent magnesium ions.

`K_concentration`**:** The molar concentration of monovalent potassium ions.

`Tris_concentration`**:** The molar concentration of the Tris(hydroxymethyl)-aminomethan buffer. Note that this value is the buffer concentration. To determine corresponding Tris ion concentration, the value of the buffer concentration is halved.

`primer_concentration`**:** The molar concentration of the PCR primers.

`template_concentration`**:** The molar concentration of the PCR templates.

For `CoverageConstraints`, a named list with constraint options. Each list entry should have an entry `min` and/or `max` in order to indicate the minimal and maximal allowed values, respectively. The following identifiers can be used as coverage constraints:

`primer_efficiency`**:** The desired efficiencies of primer-template amplification events in order to be considered as *covered*. `primer_efficiency` provides a value in the interval [0,1], which is based on **DECIPHER**'s thermodynamic model, which considers the impact of 3' terminal mismatches.

`annealing_DeltaG`**:** The desired free energies of annealing for putative coverage events between primers and templates. Typically, one would limit the maximally allowed free energy.

`stop_codon`**:** Whether coverage events introducing stop codons into the amplicons should be allowed or discarded. Here, a value of `1` indicates coverage events that induce stop codons. As such, setting both minimum and maximium to zero will disregard coverage events inducing stop codons, while setting the minimum to zero and the maximum to 1 will allow coverage events that induce stop codons.

`substitution`**:** Whether coverage events introducing substitutions into the amino acid sequence are considered or discarded. The same encoding as for `stop_codon` is used, that is, the value `1` indicates coverage events inducing substitutions. Hence, to prevent substitutions, the maximal value of `substitution` can be set to zero.

`terminal_mismatch_pos`**:** The position relative to the primer 3' terminal end for which mismatch binding events should be allowed, where the last base in a primer is indicated by position `1`. For example, setting the minimal value of `terminal_mismatch_pos` to `7` means that only coverage events that do not have a terminal mismatch within the last 6 bases of the primer are allowed.

`coverage_model`**:** Use a logistic regression model combining the free energy of annealing and 3' terminal mismatch positions to determine the expected rate of false positive coverage calls. Using `coverage_model`, you can specify the allowed ratio of falsely predicted coverage events. Typically, one would limit the maximal allowed rate of false positives. Note that setting a small false positive rate will reduce the sensitivity of the coverage calls (i.e. true positives will be missed).

For `ConstraintOptions`, a named list with constraint options. The following fields are permissible:

allowed_mismatches: The maximal number of allowed mismatches between a primer and a template sequence. If the number of mismatches of a primer with a template exceeds the specified value, the primer is not considered to cover the corresponding template when the coverage is being computed.

allowed_other_binding_ratio: Ratio of allowed binding events outside the target binding ratio. This value should be in the interval [0,1]. If the specified value is greater than zero, all coverage events outside the primer binding region are reported. If, however, the identified ratio of off-target events should exceed the allowed ratio, a warning is issued. If `allowed_other_binding_ratio` is set to `0`, only on-target primer binding events are reported. The setting of `allowed_other_binding_ratio` is ignored when designing primers, which always uses a value of 0.

allowed_region_definition: The definition of the target binding regions that is used for evaluating the coverage. In case that `allowed_region_definition` is `within`, primers have to lie within the allowed binding region. If `allowed_region_definition` is `any`, primers only have to overlap with the target binding region.

hexamer_coverage: If `hexamer_coverage` is set to "active", primers whose 3' hexamer (the last 6 bases) is fully complementary to the corresponding template region are automatically considered to cover the template. If `hexamer_coverage` is set to `inactive`, hexamer complementarity does not guarantee template coverage.

primer_coverage

Computing the primer coverage involves identifying which templates are expected to be amplified (covered) by which primers. The `primer_coverage` constraint determines the minimal and maximal number of coverage events per primer that are required. The computation of primer coverage is governed by the coverage constraints postulated via `CoverageConstraints` and the constraint options defined via `ConstraintOptions`.

primer_specificity

Primer specificity is automatically determined during the primer coverage computations but the constraint is only checked when the `primer_specificity` field is available. The specificity of a primer is defined as its ratio of on-target vs total coverage events (including off-target coverage). Low-specificity primers should be excluded as they may not amplify the target region effectively.

primer_length

The length of a primer is defined by its number of bases. Typical primers have lengths between 18 and 22. Longer primers may guarantee higher specificities.

gc_clamp

The GC clamp refers to the presence of GCs at the 3' end of a primer. For the `gc_clamp` constraint, we consider the number of 3' terminal GCs. For example, the primer *actgaaatttcaccg* has a GC clamp of length 3. The presence of a GC clamp is supposed to aid the stability of the polymerase complex. At the same time, long GC clamps should be avoided.

no_runs

Homopolymer runs (e.g. the primer *aaaaa* has a run of 5 A's) may lead to secondary structure formation and unspecific binding and should therefore be avoided.

no_repeats

Dinucleotide repeats (e.g. the primer *tatata* has 3 TA repeats) should be avoided for the same reason a long homopolymer runs.

self_dimerization

Self dimerization refers to a primer that binds to itself rather than to one of the templates. Primers exhibiting self dimers should be avoided as they may prevent the primer from amplifying the templates. Therefore primers with small free energies of dimerization should be avoided.

melting_temp_range

The melting temperature is the temperature at which 50 are in duplex with templates and 50 Hence, primers exhibiting high melting temperatures have high affinities to the templates, while primers with small melting temperatures have small affinities. The melting temperatures of the primers determine the annealing temperature of the PCR, which is why the melting temperatures of the primers should not deviate too much (see `melting_temp_diff`).

melting_temp_diff

The differences between the melting temperatures of primers in a set of primers should not deviate too much as the annealing temperaturte of a PCR should be based on the smallest melting temperature of a primer in the set. If there are other primers in the set exhibiting considerably higher melting temperatures, these primers may bind inspecifically due to the low annealing temperature.

cross_dimerization

When two different primers bind to each each other rather than to the templates, this is called cross dimerization. Cross dimerization should be prevent at all costs because such primers cannot effectively amplify their target templates. Cross dimerizing primers can be excluding primers exhibiting small free energies of cross dimerization.

secondary_structure

When a primer exhibits secondary structure, this may prevent it from binding to the templates. To prevent this, primers with low free energies of secondary structure formation can be excluded.

Note

The following external programs are required for constraint computations:

MELTING (http://www.ebi.ac.uk/biomodels/tools/melting/): Thermodynamic computations (optional) for determining melting temperatures for the constraints `melting_temp_diff` and `melting_temp_range`

OligoArrayAux (http://unafold.rna.albany.edu/OligoArrayAux.php): Thermodynamic computations used for computing `self_dimerization` and `cross_dimerization`. Also required for computing `primer_coverage` when a constraint based on the free energy of annealing is active.

ViennaRNA (http://www.tbi.univie.ac.at/RNA/): Secondary structure predictions used for the constraint `secondary_structure`

The following external programs are required for computing the coverage constraints:

OligoArrayAux (http://unafold.rna.albany.edu/OligoArrayAux.php): Thermodynamic computations used for computing the coverage constraints `annealing_DeltaG`, `primer_efficiency`, and `coverage_model`

See Also

`read_settings` for reading settings from XML files, `write_settings` for storing settings as XML files, `constraints` for accessing constraints, `constraintLimits` for accessing constraint boundaries, `cvg_constraints` for accessing coverage constraints, `conOptions` for accessing constraint options, `PCR` for accessing the PCR conditions.

Examples

```
# Initializing a new 'ConstraintSettings' object:
constraint.settings <- new("ConstraintSettings")
# Retrieving the constraint settings from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
constraints(settings)
# Modifying the constraint settings:
```

```
constraints(settings)$no_runs["max"] <- 10
constraints(settings) <- constraints(settings)[names(constraints(settings)) != "gc_clamp"]

# Initialize a new 'CoverageConstraints' object:
cvg.constraints <- new("CoverageConstraints")
# Retrieving the coverage constraints from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
cvg_constraints(settings)
# Modifying the coverage constraints
cvg_constraints(settings)$primer_efficiency["min"] <- 0.001

# Initialize a new 'ConstraintOptions' object:
constraint.options <- new("ConstraintOptions")
# Retrieve the constraint options from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
conOptions(settings)
# Prevent off-target binding:
conOptions(settings)$allowed_other_binding_ratio <- 0

# Initialize a new 'PCR_Conditions' object:
PCR.conditions <- new("PCR_Conditions")
# Retrieving the PCR conditions from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
PCR(settings)
# Modifying the PCR conditions:
PCR(settings)$use_taq_polymerase <- FALSE

# Load a settings object
filename <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(filename)
# Modify the constraints
constraints(settings)$gc_clamp["min"] <- 0
# Modify the constraint limits for designing primers
constraintLimits(settings)$gc_clamp["max"] <- 6
# Modify the coverage constraints
cvg_constraints(settings)$primer_efficiency["min"] <- 0.001
# Modify the PCR conditions
PCR(settings)$Na_concentration <- 0.0001
# Modify the constraint options
conOptions(settings)$allowed_mismatches <- 0

# Load some settings
data(Ippolito)
# View the active constraints
constraints(settings)
# Require a minimal GC clamp extent of 0
constraints(settings)$gc_clamp["min"] <- 0
# View available constraints
settings

# Load some settings
data(Ippolito)
# View all active coverage constraints
cvg_constraints(settings)
# Increase the maximal false positive rate to increase the sensitiviity of coverage predictions
cvg_constraints(settings)$coverage_model <- c("max" = 0.1)
```

```
# View available coverage constraints:
settings

# Load some settings
data(Ippolito)
# View the active constraint limits
constraintLimits(settings)
# Extend the GC relaxation limit
constraintLimits(settings)$gc_clamp <- c("min" = 0, "max" = 6)
# View available constraints
settings

# Load some settings
data(Ippolito)
# View the active PCR conditions
PCR(settings)
# Evaluate primers with a fixed annealing temperature
PCR(settings)$annealing_temperature <- 50 # celsius
# View available PCR conditions
settings

# Load some settings
data(Ippolito)
# View the active constraint options
conOptions(settings)
# Prevent mismatch binding events
conOptions(settings)$allowed_mismatches <- 0
# View available constraint options
settings
# Use two cores for parallel processing:
parallel_setup(2)
```

`TemplatesFunctions` *Template Functionalities.*

Description

`adjust_binding_regions` Adjusts the existing annotation of binding regions by specifying a new binding interval relative to the existing binding region.

`assign_binding_regions` Assigns the primer target binding regions to a set of template sequences.

`update_template_cvg` Annotates the template coverage.

`select_regions_by_conservation` Computes Shannon entropy for the defined binding regions and determines the most conserved regions.

Usage

```
update_template_cvg(template.df, primer.df, mode.directionality = NULL)

adjust_binding_regions(template.df, region.fw, region.rev)

assign_binding_regions(template.df, fw = NULL, rev = NULL,
  optimize.region = FALSE, primer.length = 20, gap.char = "-")
```

```
select_regions_by_conservation(template.df, gap.char = "-", win.len = 30,
  by.group = TRUE, mode.directionality = c("both", "fw", "rev"))
```

Arguments

`template.df`	An object of class `Templates`.
`primer.df`	An object of class `Primers` containing primers with annotated coverage that are to be used to update the template coverage in `template.df`.
`mode.directionality`	The directionality of primers/templates.
`region.fw`	Interval of new binding regions relative to the forward binding region defined in `template.df`.
`region.rev`	Interval of new binding regions relative to the reverse binding region defined in `template.df`
`fw`	Binding regions for forward primers. Either a numeric interval indicating a uniform binding range relative to the template 5' end or a path to a FASTA file providing binding sequences for every template. If `fw` is missing, only `rev` is considered.
`rev`	Binding regions for reverse primers. Either a numeric interval indicating a uniform binding range relative to the template 3' end or the path to a FASTA file providing binding sequences for every template. If `rev` is missing, only `fw` is considered.
`optimize.region`	If `TRUE`, the binding regions specified via `fw` and `rev` are adjusted such that binding regions that may form secondary structures are avoided. This feature requires ViennaRNA (see notes). If `FALSE` (the default), the input binding regions are not modified.
`primer.length`	A numeric scalar providing the probe length that is used for adjusting the primer binding regions when `optimize.region` is `TRUE`.
`gap.char`	The character in the input file representing gaps.
`win.len`	The extent of the desired primer binding region. This should be smaller than the `allowed.region`. The default is 30.
`by.group`	Shall the determination of binding regions be stratified according to the groups defined in `template.df`. By default, this is set to `TRUE`.

Details

When modifying binding regions with `adjust_binding_regions`, new binding intervals can be specified via `fw` and `rev` for forward and reverse primers, respectively. The new regions should be provided relative to the existing definition of binding regions in `template.df`. For specifying the new binding regions, position `0` refers to the first position after the end of the existing binding region. Hence, negative positions relate to regions within the existing binding region, while non-negative values relate to positions outside the defined binding region.

Binding regions are defined using `assign_binding_regions`, where the arguments `fw` and `rev` provide data describing the binding regions of the forward and reverse primers, respectively. To specify binding regions for each template individually, `fw` and `rev` should provide the paths to FASTA files. The headers of these FASTA file should match the headers of the loaded `template.df` and the sequences in the files specified by `fw` and `rev` should indicate the target binding regions.

To specify uniform binding regions, fw and rev should be numeric intervals indicating the allowed binding range for primers in the templates. The fw interval is specified with respect to the 5' end, while the rev interval is specified with respect to the template 3' end. If optimize.region is TRUE, the input binding region is adjusted such that regions forming secondary structures are avoided.

Value

update_template_cvg returns an object of class Templates with updated coverage columns.

adjust_binding_regions returns a Templates object with updated binding regions.

assign_binding_regions returns an object of class Templates with newly assigned binding regions.

select_regions_by_conservation returns a Templates object with adjusted binding regions. The attribute entropies gives a data frame with positional entropies and the attribute alignments gives the alignments of the templates.

Note

assign_binding_regions requires the program ViennaRNA (https://www.tbi.univie.ac.at/RNA/) for adjusting the binding regions when optimize.region is set to TRUE.

select_regions_by_conservation requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignme

Examples

```
# Annotate the coverage of the templates
data(Ippolito)
template.df <- update_template_cvg(template.df, primer.df)
data(Ippolito)
# Extend the binding region by one position
relative.interval <- c(-max(template.df$Allowed_End_fw), 0)
template.df.adj <- adjust_binding_regions(template.df, relative.interval)
# compare old and new annotations:
head(cbind(template.df$Allowed_Start_fw, template.df$Allowed_End_fw))
head(cbind(template.df.adj$Allowed_Start_fw, template.df.adj$Allowed_End_fw))
data(Ippolito)
# Assignment of individual binding regions
l.fasta.file <- system.file("extdata", "IMGT_data", "templates",
     "Homo_sapiens_IGH_functional_leader.fasta", package = "openPrimeR")
template.df.individual <- assign_binding_regions(template.df, l.fasta.file, NULL)
# Assign the first/last 30 bases as forward/reverse binding regions
template.df.uniform <- assign_binding_regions(template.df, c(1,30), c(1,30))
# Optimization of binding regions (requires ViennaRNA)
## Not run: template.df.opti <- assign_binding_regions(template.df, c(1,30), c(1,30),
                     optimize.region = TRUE, primer.length = 20)
## End(Not run)
data(Ippolito)
new.template.df <- select_regions_by_conservation(template.df)
```

Index