Package: glymotif (via r-universe)

May 26, 2026

Title Extract Glycan Motifs from Glycan Structures

Version 0.14.1

Description Provides comprehensive tools for glycan motif analysis and detection in glycobioinformatics research. The package enables users to identify, count, and match glycan motifs (recurring substructures) within complex glycan structures using advanced subgraph isomorphism algorithms. It includes a curated database of known motifs from the GlycoMotif GlyGen Collection, supports both concrete and generic monosaccharide matching, and offers flexible alignment options (core, terminal, or anywhere). Key functionalities include motif presence detection, occurrence counting, detailed node-to-node mapping, and batch analysis of multiple glycans against multiple motifs. The package seamlessly integrates with the glycoverse ecosystem, particularly 'glyrepr' and 'glyparse', making it essential for structural glycomics analysis, biomarker discovery, and understanding glycan-mediated biological processes.

License MIT + file LICENSE

Suggests testthat (>= 3.0.0), patrick, knitr, rmarkdown, glyexp (>= 0.12.4), glyclean (>= 0.12.0), glyread (>= 0.8.4)

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://glycoverse.github.io/glymotif/, https://github.com/glycoverse/glymotif

Imports cli, glyrepr (>= 0.10.0), glyparse (>= 0.5.4), glydraw (>= 0.4.0), igraph, purrr, rlang, stringr, checkmate, dplyr, tibble, memoise, vctrs

Depends R (>= 4.1)

VignetteBuilder knitr

BugReports https://github.com/glycoverse/glymotif/issues

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev

Repository https://glycoverse.r-universe.dev

Date/Publication 2026-05-17 02:12:25 UTC

RemoteUrl https://github.com/glycoverse/glymotif

RemoteRef v0.14.1

RemoteSha d9cebf0c9625d9420bfb765b68f21ea60cd53b3c

Contents

`add_motifs_int` *Add Motif Annotations*

Description

This function adds motif annotations to the variable information of a `glyexp::experiment()` or a tibble with a structure column. `add_motifs_int()` adds integer annotations (how many motifs are present). `add_motifs_lgl()` adds boolean annotations (whether the motif is present).

Usage

```
add_motifs_int(
  x,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)
```

```
add_motifs_lgl(
  x,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)

## S3 method for class 'glyexp_experiment'
add_motifs_int(
  x,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)

## S3 method for class 'glyexp_experiment'
add_motifs_lgl(
  x,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)

## S3 method for class 'data.frame'
add_motifs_int(
  x,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)

## S3 method for class 'data.frame'
add_motifs_lgl(
  x,
  motifs,
```

```
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  ...
)
```

Arguments

x	A glyexp::experiment() object, or a tibble with a structure column.
motifs	One of: • A glyrepr::glycan_structure() vector. • A glycan structure string vector, supported by glyparse::auto_parse(). • A character vector of motif names (use db_motifs() to see all available motifs).
alignments	A character vector specifying alignment types for each motif. Can be a single value (applied to all motifs) or a vector of the same length as motifs.
ignore_linkages	A logical value. If TRUE, linkages will be ignored in the comparison. Default is FALSE.
strict_sub	A logical value. If TRUE (default), substituents will be matched in strict mode, which means if the glycan has a substituent in some residue, the motif must have the same substituent to be matched.
match_degree	A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. For have_motif(), count_motif(), and match_motif(), this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For have_motifs(), count_motifs(), and match_motifs(), this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When match_degree is provided, alignment and alignments are silently ignored.
...	Additional arguments passed to the method.

Value

An glyexp::experiment() object with motif annotations added to the variable information.

About Names

The naming rule for the new columns follows these priorities:

1. If motifs is a named vector (character or glyrepr::glycan_structure()), the names are used directly as column names.
2. If motifs is unnamed and contains known motif names (e.g., "N-Glycan core"), the motif names are used as column names.
3. If motifs is unnamed and contains glyrepr::glycan_structure() objects or IUPAC-condensed structure strings, the IUPAC-condensed strings are used as column names.

4. If `motifs` is a motif spec from `dynamic_motifs()` or `branch_motifs()`, the IUPAC-condensed strings of the extracted motifs are used as column names.

Note: This behavior differs from `have_motifs()` and `count_motifs()`, which return matrices with NULL column names for unnamed IUPAC string or structure motifs. The functions here always provide column names since they are designed for adding motif annotations to data frames.

Why do we need these functions

Adding one motif annotation to a `glyexp::experiment()` is easy:

```
exp |>
  mutate_var(has_hex = have_motif(glycan_structure, "Hex"))
```

However, adding multiple motifs is not as straightforward. You can still use `mutate_var()` to add multiple motifs like this:

```
exp |>
  mutate_var(
    n_hex = count_motif(glycan_structure, "Hex"),
    n_dhex = count_motif(glycan_structure, "dHex"),
    n_hexnac = count_motif(glycan_structure, "HexNAc"),
  )
```

This method has two problems:

1. it has a lot of boilerplate code (a lot of typing)
2. it is not very efficient, as each call to `count_motif` performs validation and conversion on `glycan_structure`, which is a time-consuming process.

Therefore, we think it would be better to have a function that adds multiple motif annotations in a single call, in a more intuitive way. That's why we provide these two functions.

Under the hood, they use a more straightforward approach for `glyexp::experiment()` objects:

1. get the motif annotation matrix using `count_motifs()` or `have_motifs()`
2. convert the matrix to a tibble
3. use `dplyr::bind_cols()` to add the tibble to the variable information

See Also

`have_motifs()`, `count_motifs()`, `glyexp::experiment()`

Examples

```
library(glyexp)

exp <- real_experiment2

exp |>
```

```
  add_motifs_lgl(c(
    lacnac = "Gal(??-?)GlcNAc(??-",
    sia_lacnac = "Neu5Ac(??-?)Gal(??-?)GlcNAc(??-"
  )) |>
  get_var_info()

exp |>
  add_motifs_int(c(
    lacnac = "Gal(??-?)GlcNAc(??-",
    sia_lacnac = "Neu5Ac(??-?)Gal(??-?)GlcNAc(??-"
  )) |>
  get_var_info()
```

`branch_motifs` *Branch Motifs Specification*

Description

Create a specification for branch motif extraction. This should be passed to the `motifs` argument of `have_motifs()`, `count_motifs()`, `match_motifs()`, `add_motifs_lgl()`, or `add_motifs_int()`.

Usage

```
branch_motifs()
```

Details

Passing `branch_motifs()` to the `motifs` argument of supported functions will:

1. Call `extract_branch_motif()` with `including_core = TRUE` on `glycans` to get all branching motifs.
2. Construct a `match_degree` list based on the motifs.
3. Perform motif matching using the constructed `match_degree` list.

Specifically, setting `including_core = TRUE` will include an additional "Hex(??-?)Hex(??-?)HexNAc(??-?)HexNAc(??-" suffix to each branching motif. This suffix helps differentiate branching GlcNAc and bisecting GlcNAc. Then, the `match_degree` is constructed so that the four residues in the suffix do not have to match the node degree in the motif matching process.

Therefore, `have_motifs(glycans, branch_motifs())` doesn't equal to `have_motifs(glycans, extract_branch_motif(glycans))`. Never use the results from `extract_branch_motif()` directly in these functions.

Value

A `branch_motifs_spec` object.

See Also

dynamic_motifs(), extract_branch_motif()

Examples

```
glycans <- c(
  "GlcNAc(b1-2)Man(a1-3)[GlcNAc(b1-2)Man(a1-6)]Man(b1-4)GlcNAc(b1-4)GlcNAc(b1-",
  "Gal(b1-4)GlcNAc(b1-2)Man(a1-3)[Man(a1-6)]Man(b1-4)GlcNAc(b1-4)GlcNAc(b1-"
)
have_motifs(glycans, branch_motifs())
```

---

count_motif *Count How Many Times Glycans have the Given Motif(s)*

Description

These functions are closely related to have_motif(). However, instead of returning logical values, they return the number of times the glycans have the motif(s).

- count_motif() counts a single motif in multiple glycans
- count_motifs() counts multiple motifs in multiple glycans

Usage

```
count_motif(
  glycans,
  motif,
  alignment = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)

count_motifs(
  glycans,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)
```

Arguments

glycans	One of: • A glyrepr::glycan_structure() vector. • A glycan structure string vector. All formats supported by glyparse::auto_parse() are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
motif	One of: • A glyrepr::glycan_structure() scalar. • A glycan structure string, supported by glyparse::auto_parse(). • A known motif name (use db_motifs() to see all available motifs).
alignment	A character string. Possible values are "substructure", "core", "terminal", and "whole". If not provided, the value will be decided based on the motif argument. If motif is a motif name, the alignment in the database will be used. Otherwise, "substructure" will be used.
ignore_linkages	A logical value. If TRUE, linkages will be ignored in the comparison. Default is FALSE.
strict_sub	A logical value. If TRUE (default), substituents will be matched in strict mode, which means if the glycan has a substituent in some residue, the motif must have the same substituent to be matched.
match_degree	A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. For have_motif(), count_motif(), and match_motif(), this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For have_motifs(), count_motifs(), and match_motifs(), this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When match_degree is provided, alignment and alignments are silently ignored.
motifs	One of: • A glyrepr::glycan_structure() vector. • A glycan structure string vector, supported by glyparse::auto_parse(). • A character vector of motif names (use db_motifs() to see all available motifs).
alignments	A character vector specifying alignment types for each motif. Can be a single value (applied to all motifs) or a vector of the same length as motifs.

Details

This function actually perform v2f algorithm to get all possible matches between glycans and motif. However, the result is not necessarily the number of matches.

Think about the following example:

- glycan: Gal(b1-?)[Gal(b1-?)]GlcNAc(b1-4)GlcNAc(b1-
- motif: Gal(b1-?)[Gal(b1-?)]GlcNAc(b1-

To draw the glycan out:

```
Gal 1
   \ b1-? b1-4
    GlcNAc -- GlcNAc b1-
   / b1-?
Gal 2
```

To draw the motif out:

```
Gal 1
   \ b1-?
    GlcNAc b1-
   / b1-?
Gal 2
```

To differentiate the galactoses, we number them as "Gal 1" and "Gal 2" in both the glycan and the motif. The v2f subisomorphic algorithm will return two matches:

- Gal 1 in the glycan matches Gal 1 in the motif, and Gal 2 matches Gal 2.
- Gal 1 in the glycan matches Gal 2 in the motif, and Gal 2 matches Gal 1.

However, from a biological perspective, the two matches are the same. This function will take care of this, and return the "unique" number of matches.

For other details about the handling of monosaccharide, linkages, alignment, substituents, and implementation, see `have_motif()`.

Value

- `count_motif()`: An integer vector indicating how many times each `glycan` has the `motif`.
- `count_motifs()`: An integer matrix where rows correspond to glycans and columns correspond to motifs. Row names contain glycan identifiers and column names contain motif identifiers.

About Names

`have_motif()` and `count_motif()` perserve names from the input `glycans` vector.

`have_motifs()` and `count_motifs()` return a matrix with both row and column names. The row names are the glycan names, and the column names are the motif names.

Glycan names follow the same rule as `have_motif()` and `count_motif()`.

Motif names have the following rules:

1. If `motifs` have names, use the names.
2. If `motifs` don't have names and are known motif names in the database (e.g. "N-glycan core"), use them.
3. Otherwise, no colnames.

See Also

`have_motif()`, `have_motifs()`

Examples

```
library(glyparse)

count_motif("Gal(b1-3)Gal(b1-3)GalNAc(b1-", "Gal(b1-")
count_motif(
  "Man(b1-?)[Man(b1-?)]GalNAc(b1-4)GlcNAc(b1-",
  "Man(b1-?)[Man(b1-?)]GalNAc(b1-"
)
count_motif("Gal(b1-3)Gal(b1-", "Man(b1-")

# Vectorized usage with single motif
count_motif(c("Gal(b1-3)Gal(b1-3)GalNAc(b1-", "Gal(b1-3)GalNAc(b1-"), "Gal(b1-")

# Multiple motifs with count_motifs()
glycan1 <- parse_iupac_condensed("Gal(b1-3)Gal(b1-3)GalNAc(b1-")
glycan2 <- parse_iupac_condensed("Man(b1-?)[Man(b1-?)]GalNAc(b1-4)GlcNAc(b1-")
glycans <- c(glycan1, glycan2)

motifs <- c("Gal(b1-3)GalNAc(b1-", "Gal(b1-", "Man(b1-")
result <- count_motifs(glycans, motifs)
print(result)

# Monosaccharide type matching examples
# Concrete glycan vs generic motif: matches (glycan converted to generic)
count_motif("Man(?1-", "Hex(?1-") # Returns 1

# Generic glycan vs concrete motif: doesn't match
count_motif("Hex(?1-", "Man(?1-") # Returns 0
```

---

db_motifs *Get All Motifs from the Database*

Description

This function returns the names of all motifs available in the package. We use GlycoMotif GlyGen Collection (https://glycomotif.glyomics.org/glycomotif/GGM) as the source of the motifs. This function is useful to be integrated with `have_motifs()` and `count_motifs()`. For example, use `have_motifs(glycans, db_motifs())` to check against all motifs.

Usage

```
db_motifs()
```

Value

A character vector of motif names.

Examples

```
db_motifs()[1:5]
```

`dynamic_motifs` *Dynamic Motifs Specification*

Description

Create a specification for dynamic motif extraction. This should be passed to the `motifs` argument of `have_motifs()`, `count_motifs()`, `match_motifs()`, `add_motifs_lgl()`, or `add_motifs_int()`.

Usage

```
dynamic_motifs(max_size = 3)
```

Arguments

`max_size`	The maximum number of monosaccharides in the extracted motifs. Default is 3. Passed to `extract_motif()`.

Details

Passing `dynamic_motifs()` to the `motifs` argument of supported functions will:

1. Call `extract_motif()` on `glycans` to get all dynamic motifs.
2. Perform motif matching with `alignments` as "substructure".

In fact, `have_motifs(glycans, dynamic_motifs())` is just a syntatic sugar of `have_motifs(glycans, extract_motif(glycans))`. This function exists to align with the `db_motifs()` and `branch_motifs()` API.

Value

A `dynamic_motifs_spec` object.

See Also

`branch_motifs()`, `extract_motif()`

Examples

```
library(glyrepr)
glycans <- c(o_glycan_core_1(), o_glycan_core_2())
have_motifs(glycans, dynamic_motifs())
```

`extract_branch_motif` *Extract Branch Motifs*

Description

An N-glycan branching motif if the substructures linked to either the a3- or a6-core-mannose.

For example:

```
Neu5Ac - Gal - GlcNAc - Man
~~~~~~~~~~~~~~~~~~~~~      \
  A branching motif         Man - GlcNAc - GlcNAc -
                           /
                        Man
```

This function returns all the unique branching motifs found in the input glycans.

If you want to perform branching motif matching in functions like `have_motifs()` or `glydet::quantify_motifs()`, use the `branch_motifs()` helper instead, which handles additional intricacies related to how the motifs should be matched.

Usage

```
extract_branch_motif(glycans, including_core = FALSE)
```

Arguments

`glycans`	One of: • A `glyrepr::glycan_structure()` vector. • A glycan structure string vector. All formats supported by `glyparse::auto_parse()` are accepted.
`including_core`	A logical scalar. If `TRUE`, the N-glycan core structure (`Man(??-?)Man(??-?)GlcNAc(??-?)GlcNAc(??-` or `Hex(??-?)Hex(??-?)HexNAc(??-?)HexNAc(??-`) is appended to each branch motif. Default is `FALSE`.

Details

The function works by:

1. Converting the input to a set of unique `glycan_structure` objects.
2. Searching for the N-glycan branch pattern: `HexNAc(??-?)Hex(??-?)Hex(??-?)HexNAc(??-?)HexNAc(??-`.
3. For each match, identifying the root node of the branch (the leftmost HexNAc in the pattern).
4. Extracting the full subtree rooted at that node.
5. Preserving the correct anomeric configuration (e.g., "b1") by inspecting the linkage to the root node.

Value

A glyrepr::glycan_structure() vector containing the unique extracted branching motifs.

Examples

```
glycans <- c(
  "Neu5Ac(a2-3)Gal(b1-4)GlcNAc(b1-2)Man(a1-3)[Man(a1-6)]Man(b1-4)GlcNAc(a1-4)GlcNAc(b1-",
  "Gal(b1-4)GlcNAc(b1-2)Man(a1-3)[Man(a1-6)]Man(b1-4)GlcNAc(a1-4)GlcNAc(b1-"
)
extract_branch_motif(glycans)
```

---

extract_motif *Extract All Substructures (Motifs)*

Description

Extract all unique connected subgraphs (motifs) from the input glycans up to a specified size. This function can be useful combined with count_motifs() or glydet::quantify_motifs(). If so, set alignment to "substructure" for these functions.

If you want to perform dynamic motif matching in functions like have_motifs() or glydet::quantify_motifs(), use the dynamic_motifs() helper instead, which handles additional intricacies related to how the motifs should be matched.

Usage

```
extract_motif(glycans, max_size = 3)
```

Arguments

glycans	One of: • A glyrepr::glycan_structure() vector. • A glycan structure string vector. All formats supported by glyparse::auto_parse() are accepted.
max_size	The maximum number of monosaccharides in the extracted motifs. Default is 3. Note that setting this value very large can be computationally expensive. Try the default value first, and increase it progressively if needed.

Value

A glyrepr::glycan_structure() vector containing the unique extracted motifs.

Examples

```
glycan <- "Gal(b1-3)[GlcNAc(a1-6)]GalNAc(a1-"
extract_motif(glycan, max_size = 2)
```

`get_motif_structure` *Get the Structures, Alignments, or Aglycons of Known Motifs*

Description

Given a character vector of motifs names in GlycoMotif GlyGen Collection, these functions return the structures, alignments, or aglycons of the motifs.

Usage

```
get_motif_structure(name)

get_motif_alignment(name)

get_motif_aglycon(name)
```

Arguments

`name`	A character vector of the motif name.

Value

- `get_motif_structure()`: a `glyrepr::glycan_structure()`
- `get_motif_alignment()`: a character vector of motif alignments.
- `get_motif_aglycon()`: a character vector of motif aglycons.

For all three functions, if `name` has length greater than 1, the return value is named with the motif names.

See Also

`db_motifs()`, `is_known_motif()`

Examples

```
get_motif_structure("N-Glycan core basic")
get_motif_alignment("N-Glycan core basic")
get_motif_aglycon("N-Glycan core basic")

get_motif_structure(c("O-Glycan core 1", "O-Glycan core 2"))
get_motif_alignment(c("O-Glycan core 1", "O-Glycan core 2"))
get_motif_aglycon(c("O-Glycan core 1", "O-Glycan core 2"))
```

have_motif *Check if the Glycans have the Given Motif(s)*

Description

These functions check if the given `glycans` have the given `motif`(s).

- `have_motif()` checks a single motif against multiple glycans
- `have_motifs()` checks multiple motifs against multiple glycans

Technically speaking, they perform subgraph isomorphism tests to determine if the `motif`(s) are subgraphs of the `glycans`. Monosaccharides, linkages, and substituents are all considered.

Usage

```
have_motif(
  glycans,
  motif,
  alignment = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)

have_motifs(
  glycans,
  motifs,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)
```

Arguments

`glycans`	One of: • A `glyrepr::glycan_structure()` vector. • A glycan structure string vector. All formats supported by `glyparse::auto_parse()` are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
`motif`	One of: • A `glyrepr::glycan_structure()` scalar. • A glycan structure string, supported by `glyparse::auto_parse()`. • A known motif name (use `db_motifs()` to see all available motifs).
`alignment`	A character string. Possible values are "substructure", "core", "terminal", and "whole". If not provided, the value will be decided based on the `motif` argument. If `motif` is a motif name, the alignment in the database will be used. Otherwise, "substructure" will be used.

`ignore_linkages`
: A logical value. If `TRUE`, linkages will be ignored in the comparison. Default is `FALSE`.

`strict_sub`
: A logical value. If `TRUE` (default), substituents will be matched in strict mode, which means if the glycan has a substituent in some residue, the motif must have the same substituent to be matched.

`match_degree`
: A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. For `have_motif()`, `count_motif()`, and `match_motif()`, this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For `have_motifs()`, `count_motifs()`, and `match_motifs()`, this must be a list of logical vectors with length equal to `motifs`; each element follows the same length rules. When `match_degree` is provided, `alignment` and `alignments` are silently ignored.

`motifs`
: One of:

  - A `glyrepr::glycan_structure()` vector.
  - A glycan structure string vector, supported by `glyparse::auto_parse()`.
  - A character vector of motif names (use `db_motifs()` to see all available motifs).

`alignments`
: A character vector specifying alignment types for each motif. Can be a single value (applied to all motifs) or a vector of the same length as motifs.

Value

- `have_motif()`: A logical vector indicating if each `glycan` has the `motif`.
- `have_motifs()`: A logical matrix where rows correspond to glycans and columns correspond to motifs. Row names contain glycan identifiers and column names contain motif identifiers.

About Names

`have_motif()` and `count_motif()` perserve names from the input `glycans` vector.

`have_motifs()` and `count_motifs()` return a matrix with both row and column names. The row names are the glycan names, and the column names are the motif names.

Glycan names follow the same rule as `have_motif()` and `count_motif()`.

Motif names have the following rules:

1. If `motifs` have names, use the names.
2. If `motifs` don't have names and are known motif names in the database (e.g. "N-glycan core"), use them.
3. Otherwise, no colnames.

Monosaccharide type

As of glyrepr 0.9.0.9000, all elements in a `glycans` or `motifs` vector must have the same monosaccharide type ("concrete" or "generic"). This invariant is enforced when creating or combining `glyrepr_structure` objects. The matching rules are:

- When the motif is "generic", glycans are converted to "generic" type for comparison, allowing both concrete and generic glycans to match generic motifs.
- When the motif is "concrete", glycans are used as-is, so only concrete glycans with matching monosaccharide names will match, while generic glycans will not match.

Examples:

- `Man` (concrete glycan) vs `Hex` (generic motif) → TRUE (Man converted to Hex for comparison)
- `Hex` (generic glycan) vs `Man` (concrete motif) → FALSE (names don't match)
- `Man` (concrete glycan) vs `Man` (concrete motif) → TRUE (exact match)
- `Hex` (generic glycan) vs `Hex` (generic motif) → TRUE (exact match)

Linkages

Obscure linkages (e.g. "??-?") are allowed in the `motif` graph (see `glyrepr::possible_linkages()`). "?" in a motif graph means "anything could be OK", so it will match any linkage in the `glycan` graph. However, "?" in a `glycan` graph will only match "?" in the `motif` graph. You can set `ignore_linkages = TRUE` to ignore linkages in the comparison.

Some examples:

- "b1-?" in motif will match "b1-4" in glycan.
- "b1-?" in motif will match "b1-?" in glycan.
- "b1-4" in motif will NOT match "b1-?" in glycan.
- "a1-?" in motif will NOT match "b1-4" in glycan.
- "a1-?" in motif will NOT match "a?-4" in glycan.

Both motifs and glycans can have a "half-linkage" at the reducing end, e.g. "GlcNAc(b1-". The half linkage in the motif will be matched to any linkage in the glycan, or the half linkage of the glycan. e.g. Glycan "GlcNAc(b1-4)Gal(a1-" will have both "GlcNAc(b1-" and "Gal(a1-" motifs.

Alignment

According to the GlycoMotif database, a motif can be classified into four alignment types:

- "substructure": The motif can be anywhere in the glycan. This is the default. See substructure for details.
- "core": The motif must align with at least one connected substructure (subtree) at the reducing end of the glycan. See glycan core for details.
- "terminal": The motif must align with at least one connected substructure (subtree) at the nonreducing end of the glycan. See nonreducing end for details.
- "whole": The motif must align with the entire glycan. See whole-glycan for details.

When using known motifs in the GlycoMotif GlyGen Collection, the best practice is to not provide the `alignment` argument, and let the function decide the alignment based on the motif name. However, it is still possible to override the default alignments. In this case, the user-provided alignments will be used, but a warning will be issued. When `match_degree` is provided, `alignment` and `alignments` are ignored without warning.

Degree matching

`match_degree` is used to require exact degree matching for specific motif nodes. For each node marked `TRUE`, the matched glycan node must have the same in-degree and out-degree as the motif node. Nodes marked `FALSE` do not enforce degree equality. This is useful to prevent matches where the motif node is embedded in a more highly branched glycan region (extra outgoing edges) or has extra incoming connections compared to the motif.

Substituents

Substituents (e.g. "Ac", "SO3") are matched in strict mode. Both single and multiple substituents are supported:

- Single substituents: "Neu5Ac-9Ac" will only match "Neu5Ac-9Ac" but not "Neu5Ac"
- Multiple substituents: "Glc3Me6S" (has both 3Me and 6S) will only match motifs that contain both substituents, e.g., "Glc3Me6S", "Glc?Me6S", "Glc3Me?S"
- "Glc3Me6S" will NOT match "Glc3Me" (missing 6S) or "Glc" (missing both)

For multiple substituents, they are internally stored as comma-separated values (e.g. "3Me,6S") and matched individually. Each substituent in the motif must have a corresponding match in the glycan, and vice versa.

Obscure linkages in motif substituents will match any linkage in glycan substituents:

- Motif "Neu5Ac?Ac" will match "Neu5Ac9Ac" in the glycan
- Motif "Glc?Me6S" will match "Glc3Me6S" in the glycan (? matches 3)
- Motif "Glc3Me?S" will match "Glc3Me6S" in the glycan (? matches 6)

Fuzzy built-in residue modifications in motifs also match fully specified target glycans. For example, motif "Gal?NAc" matches glycan "GalNAc", and motif "Neu?Ac" matches glycan "Neu5Ac".

This default behavior is reasonable for most cases, because monosaccharides with different substituents should be regarded as different. However, you can change this behavior by setting `strict_sub = FALSE`. In this case, the substituent is optional in the motif, so the glycan "Neu5Ac9Ac" can match the motif "Neu5Ac".

Implementation

Under the hood, the function uses `igraph::graph.get.subisomorphisms.vf2()` to get all possible subgraph isomorphisms between `glycan` and `motif`. `color` vertex attributes are added to the graphs to distinguish monosaccharides. For all possible matches, the function checks the following:

- Alignment: using `alignment_check()`
- Residues and substituents: using `residue_check()`
- Degree: using `degree_check()` (only when `match_degree` is provided)
- Linkages: using `linkage_check()`
- Anomer: using `anomer_check()` The function returns `TRUE` if any of the matches pass all checks.

See Also

count_motif(), count_motifs(), glyparse::auto_parse()

Examples

```
library(glyparse)
library(glyrepr)

(glycan <- o_glycan_core_2(mono_type = "concrete"))

# The glycan has the motif "Gal(b1-3)GalNAc(b1-"
have_motif(glycan, "Gal(b1-3)GalNAc(b1-")

# But not "Gal(b1-4)GalNAc(b1-" (wrong linkage)
have_motif(glycan, "Gal(b1-4)GalNAc(b1-")

# Set `ignore_linkages` to `TRUE` to ignore linkages
have_motif(glycan, "Gal(b1-4)GalNAc(b1-", ignore_linkages = TRUE)

# Different monosaccharide types are allowed
have_motif(glycan, "Hex(b1-3)HexNAc(?1-")

# Obscure linkages in the `motif` graph are allowed
have_motif(glycan, "Gal(b1-?)GalNAc(?1-")

# However, obscure linkages in `glycan` will only match "?" in the `motif` graph
glycan_2 <- parse_iupac_condensed("Gal(b1-?)[GlcNAc(b1-6)]GalNAc(?1-")
have_motif(glycan_2, "Gal(b1-3)GalNAc(?1-")
have_motif(glycan_2, "Gal(b1-?)GalNAc(?1-")

# The anomer of the motif will be matched to linkages in the glycan
have_motif(glycan_2, "GlcNAc(b1-")

# Alignment types
# The default type is "substructure", which means the motif can be anywhere in the glycan.
# Other options include "core", "terminal" and "whole".
glycan_3 <- parse_iupac_condensed("Gal(a1-3)Gal(a1-4)Gal(a1-6)Gal(a1-")
motifs <- c(
  "Gal(a1-3)Gal(a1-4)Gal(a1-6)Gal(a1-",
  "Gal(a1-3)Gal(a1-4)Gal(a1-",
  "Gal(a1-4)Gal(a1-6)Gal(a1-",
  "Gal(a1-4)Gal(a1-"
)

purrr::map_lgl(motifs, ~ have_motif(glycan_3, .x, alignment = "whole"))
purrr::map_lgl(motifs, ~ have_motif(glycan_3, .x, alignment = "core"))
purrr::map_lgl(motifs, ~ have_motif(glycan_3, .x, alignment = "terminal"))
purrr::map_lgl(motifs, ~ have_motif(glycan_3, .x, alignment = "substructure"))

# Substituents
glycan_4 <- "Neu5Ac9Ac(a2-3)Gal(b1-4)GlcNAc(b1-"
glycan_5 <- "Neu5Ac(a2-3)Gal(b1-4)GlcNAc(b1-"
```

```
have_motif(glycan_4, glycan_5)
have_motif(glycan_5, glycan_4)
have_motif(glycan_4, glycan_4)
have_motif(glycan_5, glycan_5)

have_motif(glycan_4, glycan_5, strict_sub = FALSE)
have_motif(glycan_5, glycan_4, strict_sub = FALSE)
have_motif(glycan_4, glycan_4, strict_sub = FALSE)
have_motif(glycan_5, glycan_5, strict_sub = FALSE)

# Multiple substituents
glycan_6 <- "Glc3Me6S(a1-" # has both 3Me and 6S substituents
have_motif(glycan_6, "Glc3Me6S(a1-") # TRUE: exact match
have_motif(glycan_6, "Glc?Me6S(a1-") # TRUE: obscure linkage ?Me matches 3Me
have_motif(glycan_6, "Glc3Me?S(a1-") # TRUE: obscure linkage ?S matches 6S
have_motif(glycan_6, "Glc3Me(a1-") # FALSE: missing 6S substituent
have_motif(glycan_6, "Glc(a1-") # FALSE: missing all substituents

# Vectorization with single motif
glycans <- c(glycan, glycan_2, glycan_3)
motif <- "Gal(b1-3)GalNAc(b1-"
have_motif(glycans, motif)

# Multiple motifs with have_motifs()
glycan1 <- o_glycan_core_2(mono_type = "concrete")
glycan2 <- parse_iupac_condensed("Gal(b1-?)[GlcNAc(b1-6)]GalNAc(b1-")
glycans <- c(glycan1, glycan2)

motifs <- c("Gal(b1-3)GalNAc(b1-", "Gal(b1-4)GalNAc(b1-", "GlcNAc(b1-6)GalNAc(b1-")
have_motifs(glycans, motifs)

# You can assign each motif a name
motifs <- c(
  motif1 = "Gal(b1-3)GalNAc(b1-",
  motif2 = "Gal(b1-4)GalNAc(b1-",
  motif3 = "GlcNAc(b1-6)GalNAc(b1-"
)
have_motifs(glycans, motifs)
```

`is_known_motif` *Check if a Motif is Known*

Description

This function checks if motifs are known motifs in GlycoMotif GlyGen Collection.

Usage

```
is_known_motif(name)
```

Arguments

name	A character vector of the motif name.

Value

A logical vector.

Examples

```
is_known_motif(c("N-Glycan core basic", "O-Glycan core 1", "unknown"))
```

---

match_motif *Match Motif(s) in Glycans*

Description

These functions find all occurrences of the given motif(s) in the glycans. Node-to-node mapping is returned for each match. This function is NOT useful for most users if you are not interested in the concrete node mapping. See have_motif() and count_motif() for more information about the matching rules.

- match_motif() matches a single motif against multiple glycans
- match_motifs() matches multiple motifs against multiple glycans

Different from have_motif() and count_motif(), these functions return detailed match information. More specifically, for each glycan-motif pair, a integer vector is returned, indicating the node mapping from the motif to the glycan. For example, if the vector is c(2, 3, 6), it means that the first node in the motif matches the 2nd node in the glycan, the second node in the motif matches the 3rd node in the glycan, and the third node in the motif matches the 6th node in the glycan.

Node indices are only meaningful for glyrepr::glycan_structure(), so only glyrepr::glycan_structure() is supported for glycans and motifs.

Usage

```
match_motif(
  glycans,
  motif,
  alignment = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)

match_motifs(
  glycans,
  motifs,
  alignments = NULL,
```

```
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)
```

Arguments

`glycans`	One of: • A `glyrepr::glycan_structure()` vector. • A glycan structure string vector. All formats supported by `glyparse::auto_parse()` are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
`motif`	One of: • A `glyrepr::glycan_structure()` scalar. • A glycan structure string, supported by `glyparse::auto_parse()`. • A known motif name (use `db_motifs()` to see all available motifs).
`alignment`	A character string. Possible values are "substructure", "core", "terminal", and "whole". If not provided, the value will be decided based on the `motif` argument. If `motif` is a motif name, the alignment in the database will be used. Otherwise, "substructure" will be used.
`ignore_linkages`	A logical value. If `TRUE`, linkages will be ignored in the comparison. Default is `FALSE`.
`strict_sub`	A logical value. If `TRUE` (default), substituents will be matched in strict mode, which means if the glycan has a substituent in some residue, the motif must have the same substituent to be matched.
`match_degree`	A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. For `have_motif()`, `count_motif()`, and `match_motif()`, this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For `have_motifs()`, `count_motifs()`, and `match_motifs()`, this must be a list of logical vectors with length equal to `motifs`; each element follows the same length rules. When `match_degree` is provided, `alignment` and `alignments` are silently ignored.
`motifs`	One of: • A `glyrepr::glycan_structure()` vector. • A glycan structure string vector, supported by `glyparse::auto_parse()`. • A character vector of motif names (use `db_motifs()` to see all available motifs).
`alignments`	A character vector specifying alignment types for each motif. Can be a single value (applied to all motifs) or a vector of the same length as motifs.

Value

A nested list of integer vectors.

- `match_motif()`: Two levels of nesting. The outer list corresponds to glycans, and the inner list corresponds to matches. Use `purrr::pluck(result, glycan_index, match_index)` to access the match information. For example, `purrr::pluck(result, 1, 2)` means the 2nd match in the 1st glycan.
- `match_motifs()`: Three levels of nesting. The outermost list corresponds to motifs, the middle list corresponds to glycans, and the innermost list corresponds to matches. Use `purrr::pluck(result, motif_index, glycan_index, match_index)` to access the match information. For example, `purrr::pluck(result, 1, 2, 3)` means the 3rd match in the 2nd glycan for the 1st motif. The outermost list is named by `motifs` if they have names. The middle list is named by `glycans` if they have names.

Vertex and Linkage Indices

The indices of vertices and linkages in a glycan correspond directly to their order in the IUPAC-condensed string, which is printed when you print a `glyrepr::glycan_structure()`. For example, for the glycan `Man(a1-3)[Man(a1-6)]Man(b1-4)GlcNAc(b1-4)GlcNAc(b1-)`, the vertices are "Man", "Man", "Man", "GlcNAc", "GlcNAc", and the linkages are "a1-3", "a1-6", "b1-4", "b1-4".

Thus, matching the motif "Man(a1-3)Man(b1-4)" to this glycan yields `c(1, 3)`. This indicates that the first motif vertex (the a1-3 Man) corresponds to the first vertex in the glycan, and the second motif vertex (the b1-4 Man) corresponds to the third vertex in the glycan.

About Names

`match_motif()` perserve names from the input `glycans` vector.‘

For `match_motifs()`, the outermost list is named by motifs, and the inner lists are named by glycans, following the same rules as in `have_motifs()` and `count_motifs()`.

See Also

`have_motif()`, `count_motif()`

Examples

```
library(glyparse)
library(glyrepr)

(glycan <- n_glycan_core())

# Let's peek under the hood of the nodes in the glycan
glycan_graph <- get_structure_graphs(glycan)
igraph::V(glycan_graph)$mono # 1, 2, 3, 4, 5

# Match a single motif against a single glycan
motif <- parse_iupac_condensed("Man(a1-3)[Man(a1-6)]Man(b1-")
match_motif(glycan, motif)

# Match multiple motifs against a single glycan
motifs <- c(
  "Man(a1-3)[Man(a1-6)]Man(b1-",
```

```
  "Man(a1-3)Man(b1-4)GlcNAc(b1-4)GlcNAc(?1-"
)
motifs <- parse_iupac_condensed(motifs)
match_motifs(glycan, motifs)
```

`view_motif` *View motif matches on a glycan*

Description

Visualize where a motif matches a glycan structure.

Usage

```
view_motif(
  glycan,
  motif,
  alignment = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL
)
```

Arguments

`glycan`	One of: • A scalar `glyrepr::glycan_structure()`. • A glycan structure string. All formats supported by `glyparse::auto_parse()` are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
`motif`	One of: • A `glyrepr::glycan_structure()` scalar. • A glycan structure string, supported by `glyparse::auto_parse()`. • A known motif name (use `db_motifs()` to see all available motifs).
`alignment`	A character string. Possible values are "substructure", "core", "terminal", and "whole". If not provided, the value will be decided based on the `motif` argument. If `motif` is a motif name, the alignment in the database will be used. Otherwise, "substructure" will be used.
`ignore_linkages`	A logical value. If `TRUE`, linkages will be ignored in the comparison. Default is `FALSE`.
`strict_sub`	A logical value. If `TRUE` (default), substituents will be matched in strict mode, which means if the glycan has a substituent in some residue, the motif must have the same substituent to be matched.

match_degree A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. For have_motif(), count_motif(), and match_motif(), this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For have_motifs(), count_motifs(), and match_motifs(), this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When match_degree is provided, alignment and alignments are silently ignored.

Details

view_motif() matches one motif against one glycan with the same matching rules used by match_motif(), then draws the glycan with the matched residues highlighted.

Value

A ggplot object returned by glydraw::draw_cartoon(). If no match is found, the glycan is drawn without highlighted residues and a cli alert is emitted.

See Also

match_motif(), glydraw::draw_cartoon()

Examples

```
library(glyparse)
library(glyrepr)

glycan <- n_glycan_core()
motif <- parse_iupac_condensed("Man(a1-3)[Man(a1-6)]Man(b1-")

## Not run:
view_motif(glycan, motif)

## End(Not run)
```

Index