

Package ‘glymotif’

October 7, 2026

Title Extract Glycan Motifs from Glycan Structures

Version 1.0.0

Description Identify, count, and match recurring substructures in glycan structures. Supports concrete and generic monosaccharide matching, several structural alignment modes, node-to-node mappings, and batch analysis using subgraph isomorphism. Includes curated motif annotations derived from the 'GlycoMotif' resource <<https://glycomotif.glyomics.org/>>. Integrates with 'glyrepr' and 'glyparse' for structural glycomics workflows.

License MIT + file LICENSE

Suggests testthat (>= 3.0.0), patrick, knitr, rmarkdown, dplyr

Config/testthat/edition 3

Encoding UTF-8

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

Imports cli, glyrepr (>= 1.0.0), glyparse (>= 0.6.0), glydraw (>= 0.4.0), igraph, purrr, rlang, stringr, checkmate, tibble, vctrs, lifecycle, Rcpp

LinkingTo BH, Rcpp

Depends R (>= 4.1)

VignetteBuilder knitr

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

RoxygenNote 7.3.3

NeedsCompilation yes

Author Bin Fu [aut, cre, cph] (ORCID: <<https://orcid.org/0000-0001-8567-2997>>)

Maintainer Bin Fu <23110220018@m.fudan.edu.cn>

Repository CRAN

Date/Publication 2026-10-07 08:30:15 UTC

Contents

branch_motifs	2
count_motif	3
db_motifs	7
db_motif_info	8
dot-g_motif	9
dynamic_motifs	11
extract_branch_motif	12
extract_motif	13
get_motif_structure	14
have_motif	15
is_known_motif	21
match_motif	22
view_motif	25
Index	28

branch_motifs	<i>Branch Motifs Specification</i>
---------------	------------------------------------

Description

Create a specification for branch motif extraction. This should be passed to the motifs argument of [have_motifs\(\)](#), [count_motifs\(\)](#), or [match_motifs\(\)](#).

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,
  mode = c("strict", "lenient"),
  strict_floating = TRUE
)
```

```
count_motifs(
  glycans,
  motifs,
  ...,
  alignments = NULL,
  ignore_linkages = FALSE,
```

```

    strict_sub = TRUE,
    match_degree = NULL,
    mode = c("strict", "lenient"),
    strict_floating = TRUE
  )

```

Arguments

glycans	One of: - A <code>glyrepr::glycan_structure()</code> vector. - A glycan structure string vector. All formats supported by <code>glyparse::auto_parse()</code> are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
motif	One of: - A <code>glyrepr::glycan_structure()</code> scalar. - A glycan structure string, supported by <code>glyparse::auto_parse()</code>. - A GGM database motif name (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
alignment	A character string. Possible values are "substructure", "core", "terminal", and "whole". If not provided, the value will be decided based on the <code>motif</code> argument. If <code>motif</code> is a GGM 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 <code>have_motif()</code> , <code>count_motif()</code> , and <code>match_motif()</code> , this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For <code>have_motifs()</code> , <code>count_motifs()</code> , and <code>match_motifs()</code> , this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When <code>match_degree</code> is provided, <code>alignment</code> and <code>alignments</code> are silently ignored.
mode	Matching mode. "strict" preserves the default behavior where glycans cannot be more obscure than motifs. "lenient" treats glycan-side obscure fields as compatible with more specific motif fields while still rejecting concrete mismatches.
strict_floating	A logical value. If TRUE (default), a motif is present only when it occurs in every conflict-free localization of any floating glycan parts or substituents. If FALSE, a motif is present when it occurs in at least one possible localization.
motifs	One of:

- A `glyrepr::glycan_structure()` vector.
 - A glycan structure string vector, supported by `glyparse::auto_parse()`.
 - A character vector of GGM database motif names (use `db_motif_info()` |> `dplyr::filter(source_id == "GGM")` to inspect resolvable motif names).
- 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() preserve 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 GGM database motif names (e.g. "N-glycan core"), use them.
3. Otherwise, no colnames.

Floating parts and substituents

Glycans with unresolved floating parts or substituents are matched across every conflict-free localization allowed by their candidate-parent domains. strict_floating = TRUE requires a match in every localization, while strict_floating = FALSE requires a match in at least one localization. This setting is independent of mode, which controls residue and linkage obscurity.

count_motif() and count_motifs() return the minimum count across localizations in strict-floating mode and the maximum count otherwise. match_motif() and match_motifs() return the union of node mappings from every localization, using node indices from the original unresolved structure.

Motifs must be connected structures and therefore cannot themselves contain unresolved floating parts or substituents.

Matching supports up to 256 raw candidate-parent combinations per glycan. Localize floating parts and substituents with glyrepr::localize_floating_parts() first for larger domains.

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: compatible residues match
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 a database motif specification. We use GlycoMotif collections (<https://glycomotif.glycomics.org/glycomotif>) 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 the default GlyGen motif collection, or pass `source_id` to use another collection.

Usage

```
db_motifs(source_id = "GGM")
```

Arguments

<code>source_id</code>	A character vector of motif collection identifiers to use. Defaults to "GGM" for backward compatibility. Use <code>dplyr::distinct(db_motif_info(), source_id, source)</code> to get all available sources. You can use more than one motif collections like <code>c("GGM", "CCRC")</code> . To use all available motifs, use the "GM" collection directly.
------------------------	---

Details

Use `db_motif_info()` to inspect the motifs included in the database. You can use `dplyr::distinct(db_motif_info(), source_id, source)` to get all available sources.

Value

A `db_motifs_spec` object.

Data source and license

The bundled annotations are derived from the **GlycoMotif** resource. GlyGen distributes its database sets under the **Creative Commons Attribution 4.0 International license**. See the package COPYRIGHTS file for attribution and snapshot details.

Examples

```
db_motifs()
```

db_motif_info	<i>Get Database Motif Information</i>
---------------	---------------------------------------

Description

Returns metadata for all motifs available in the package. You can use `dplyr::distinct(db_motif_info(), source_id, source)` to get all available sources.

Usage

```
db_motif_info()
```

Details

It contains the following columns:

- `source_id`: the collection identifier of the motif
- `source`: the collection name of the motif
- `accession`: the accession number of the motif
- `name`: the name of the motif
- `alignment`: the alignment of the motif
- `glycan_structure`: the glycan structure (`glyrepr::glycan_structure()`) of the motif

Value

A tibble.

Examples

```
db_motif_info()
```

Description

These functions are low-level variants of `have_motif()`, `count_motif()`, and `match_motif()` for package code that already has compatible igraph objects from `glyrepr::get_structure_graphs()`.

Usage

```
.g_have_motif(  
  glycan_graph,  
  motif_graph,  
  ...,  
  alignment = "substructure",  
  ignore_linkages = FALSE,  
  strict_sub = TRUE,  
  match_degree = NULL,  
  mode = c("strict", "lenient"),  
  strict_floating = TRUE  
)  
  
.g_count_motif(  
  glycan_graph,  
  motif_graph,  
  ...,  
  alignment = "substructure",  
  ignore_linkages = FALSE,  
  strict_sub = TRUE,  
  match_degree = NULL,  
  mode = c("strict", "lenient"),  
  strict_floating = TRUE  
)  
  
.g_match_motif(  
  glycan_graph,  
  motif_graph,  
  ...,  
  alignment = "substructure",  
  ignore_linkages = FALSE,  
  strict_sub = TRUE,  
  match_degree = NULL,  
  mode = c("strict", "lenient")  
)
```

Arguments

`glycan_graph` An igraph glycan graph.

motif_graph	An igraph motif graph.
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
alignment	A character scalar: "substructure", "core", "terminal", or "whole".
ignore_linkages	A logical scalar. If TRUE, linkages are ignored.
strict_sub	A logical scalar. If TRUE, substituents are matched strictly.
match_degree	A logical vector indicating which motif nodes must match the glycan's in- and out-degree exactly. A scalar is recycled to the number of motif nodes.
mode	Matching mode. "strict" preserves the default behavior; "lenient" treats glycan-side unknowns as compatible with more specific motif fields.
strict_floating	A logical scalar. For .g_have_motif(), TRUE requires the motif in every possible floating localization and FALSE requires it in at least one. For .g_count_motif(), TRUE returns the minimum count across localizations and FALSE returns the maximum.

Details

These functions do no validation, parsing, naming, or graph mutation. Callers must provide valid graph objects. Residue compatibility follows the high-level matching rules, including generic and mixed motif residues.

These functions never call `glyrepr::as_glycan_structure()`.

Glycan graphs with unresolved floating parts or substituents are matched across all conflict-free localizations. `.g_match_motif()` returns the union of mappings from every localization, with node indices referring to the original unresolved graph.

Value

- `.g_have_motif()` returns a logical scalar.
- `.g_count_motif()` returns an integer scalar.
- `.g_match_motif()` returns a list of integer vectors.

See Also

`have_motif()`, `count_motif()`, `match_motif()`

Examples

```
library(glyparse)
library(glyrepr)

glycan <- parse_iupac_condensed("Gal(b1-3)GalNAc(b1-")
motif <- parse_iupac_condensed("Gal(b1-")
glycan_graph <- get_structure_graphs(glycan)
motif_graph <- get_structure_graphs(motif)
```

```
.g_have_motif(glycan_graph, motif_graph)
.g_count_motif(glycan_graph, motif_graph)
.g_match_motif(glycan_graph, motif_graph)
```

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\(\)](#), or [match_motifs\(\)](#).

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

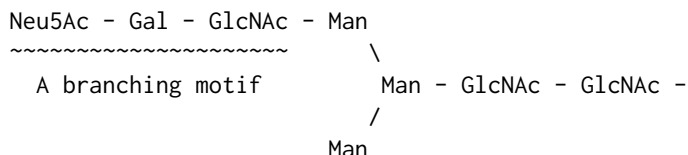
```
library(glyrepr)
glycans <- c(o_glycan_core_1(), o_glycan_core_2())
have_motifs(glycans, dynamic_motifs())
```

 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:



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.
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
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. Concrete branches receive a concrete core; generic and mixed branches receive a generic core. 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 <code>glyrepr::glycan_structure()</code> vector. - A glycan structure string vector. All formats supported by <code>glyparse::auto_parse()</code> are accepted.
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
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	<i>Get the Structures or Alignments of Known Motifs</i>
---------------------	---

Description

[Deprecated]

get_motif_structure() and get_motif_alignment() were deprecated in glymotif 0.16.0. Use [db_motif_info\(\)](#) to inspect database motifs instead.

Usage

```
get_motif_structure(name)
```

```
get_motif_alignment(name)
```

Arguments

name	A character vector of motif names.
------	------------------------------------

Value

- get_motif_structure(): a [glyrepr::glycan_structure\(\)](#) vector.
- get_motif_alignment(): a character vector of motif alignments.

For get_motif_alignment(), if name has length greater than 1, the return value is named with the motif names.

See Also

[db_motif_info\(\)](#)

Examples

```
get_motif_structure("LacdiNAc")  
get_motif_alignment("LacdiNAc")  
  
get_motif_structure(c("O-Glycan core 1", "O-Glycan core 2"))  
get_motif_alignment(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,
  mode = c("strict", "lenient"),
  strict_floating = TRUE
)
```

```
have_motifs(
  glycans,
  motifs,
  ...,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  mode = c("strict", "lenient"),
  strict_floating = TRUE
)
```

Arguments

glycans	One of: • A <code>glyrepr::glycan_structure()</code> vector. • A glycan structure string vector. All formats supported by <code>glyparse::auto_parse()</code> are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
motif	One of:

	• A <code>glyrepr::glycan_structure()</code> scalar. • A glycan structure string, supported by <code>glyparse::auto_parse()</code>. • A GGM database motif name (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
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 GGM 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 <code>have_motif()</code> , <code>count_motif()</code> , and <code>match_motif()</code> , this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For <code>have_motifs()</code> , <code>count_motifs()</code> , and <code>match_motifs()</code> , this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When <code>match_degree</code> is provided, alignment and alignments are silently ignored.
mode	Matching mode. "strict" preserves the default behavior where glycans cannot be more obscure than motifs. "lenient" treats glycan-side obscure fields as compatible with more specific motif fields while still rejecting concrete mismatches.
strict_floating	A logical value. If TRUE (default), a motif is present only when it occurs in every conflict-free localization of any floating glycan parts or substituents. If FALSE, a motif is present when it occurs in at least one possible localization.
motifs	One of: • A <code>glyrepr::glycan_structure()</code> vector. • A glycan structure string vector, supported by <code>glyparse::auto_parse()</code>. • A character vector of GGM database motif names (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
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() preserve 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 GGM database motif names (e.g. "N-glycan core"), use them.
3. Otherwise, no colnames.

Monosaccharide type

Glycans and motifs can each contain concrete residues, generic residues, or a mixture of both. Structure vectors can likewise combine concrete, generic, and mixed elements. Matching is performed residue by residue for every glycan-motif pair; structures are not converted as a whole.

In the default strict mode:

- Concrete glycan residues match the same concrete motif residue.
- Concrete glycan residues also match compatible generic motif residues.
- Generic glycan residues do not match concrete motif residues.
- Generic glycan residues match the same generic motif residue.
- Concrete residues with different identities never match.

Examples:

- Man (concrete glycan) vs Hex (generic motif) → TRUE
- Hex (generic glycan) vs Man (concrete motif) → FALSE
- Man (concrete glycan) vs Man (concrete motif) → TRUE
- Hex (generic glycan) vs Hex (generic motif) → TRUE
- Gal(b1-3)GalNAc (concrete glycan) vs Hex(b1-3)GalNAc (mixed motif) → TRUE

With mode = "lenient", compatibility becomes bidirectional: generic glycan residues can also match compatible concrete motif residues. For example, Hex can match a Gal motif residue, but HexNAc still cannot match Gal.

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.

Matching mode

mode = "strict" is the default and preserves the standard rule that glycans cannot be more obscure than motifs. For example, glycan "Gal(?1-?)GalNAc(?1-" does not match motif "Gal(b1-3)GalNAc(a1-".

mode = "lenient" treats obscure glycan-side monosaccharides, linkages, substituent positions, and reducing-end anomers as compatible with more specific motif fields. In the lenient mode, glycan "Gal(?1-?)GalNAc(?1-" matches motif "Gal(b1-3)GalNAc(a1-". Concrete mismatches still fail: for example, glycan "Gal(?1-6)GalNAc(a1-" does not match motif "Gal(b1-3)GalNAc(a1-".

Floating parts and substituents

Glycans with unresolved floating parts or substituents are matched across every conflict-free localization allowed by their candidate-parent domains. `strict_floating = TRUE` requires a match in every localization, while `strict_floating = FALSE` requires a match in at least one localization. This setting is independent of mode, which controls residue and linkage obscurity.

`count_motif()` and `count_motifs()` return the minimum count across localizations in strict-floating mode and the maximum count otherwise. `match_motif()` and `match_motifs()` return the union of node mappings from every localization, using node indices from the original unresolved structure.

Motifs must be connected structures and therefore cannot themselves contain unresolved floating parts or substituents.

Matching supports up to 256 raw candidate-parent combinations per glycan. Localize floating parts and substituents with `glyrepr::localize_floating_parts()` first for larger domains.

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 named motifs in the GlycoMotif GlyGen Collection (GGM), 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. Only GGM motif names are accepted as character name inputs through the `motif` and `motifs` arguments. To use motifs from other database collections, pass `db_motifs()` with the desired `source_id` or pass structures from `db_motif_info()` explicitly. 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 Boost Graph's VF2 subgraph monomorphism algorithm. Custom vertex and edge compatibility predicates enforce residue, substituent, linkage, anomer, alignment, and degree constraints during the search. The function returns `TRUE` as soon as a compatible mapping is found.

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

[Deprecated]

is_known_motif() was deprecated in glymotif 0.16.0. Use [db_motif_info\(\)](#) to inspect database motifs instead.

Usage

```
is_known_motif(name)
```

Arguments

name A character vector of motif names.

Value

A logical vector.

Examples

```
is_known_motif(c("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,
  mode = c("strict", "lenient")
```

```

)

match_motifs(
  glycans,
  motifs,
  ...,
  alignments = NULL,
  ignore_linkages = FALSE,
  strict_sub = TRUE,
  match_degree = NULL,
  mode = c("strict", "lenient")
)

```

Arguments

glycans	One of: - A <code>glyrepr::glycan_structure()</code> vector. - A glycan structure string vector. All formats supported by <code>glyparse::auto_parse()</code> are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
motif	One of: - A <code>glyrepr::glycan_structure()</code> scalar. - A glycan structure string, supported by <code>glyparse::auto_parse()</code>. - A GGM database motif name (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
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 GGM 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 <code>have_motif()</code> , <code>count_motif()</code> , and <code>match_motif()</code> , this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For <code>have_motifs()</code> , <code>count_motifs()</code> , and <code>match_motifs()</code> , this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When <code>match_degree</code> is provided, <code>alignment</code> and <code>alignments</code> are silently ignored.
mode	Matching mode. "strict" preserves the default behavior where glycans cannot be more obscure than motifs. "lenient" treats glycan-side obscure fields as compatible with more specific motif fields while still rejecting concrete mismatches.

motifs	One of: • A <code>glyrepr::glycan_structure()</code> vector. • A glycan structure string vector, supported by <code>glyparse::auto_parse()</code>. • A character vector of GGM database motif names (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
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()` preserve 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()`.

Floating parts and substituents

Glycans with unresolved floating parts or substituents are matched across every conflict-free localization allowed by their candidate-parent domains. These functions return the union of node mappings from every localization, using node indices from the original unresolved structure.

Motifs must be connected structures and therefore cannot themselves contain unresolved floating parts or substituents.

Matching supports up to 256 raw candidate-parent combinations per glycan. Localize floating parts and substituents with `glyrepr::localize_floating_parts()` first for larger domains.

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 <code>glyrepr::glycan_structure()</code>. - A glycan structure string. All formats supported by <code>glyparse::auto_parse()</code> are accepted, including IUPAC-condensed, WURCS, GlycoCT, and others.
motif	One of: - A <code>glyrepr::glycan_structure()</code> scalar. - A glycan structure string, supported by <code>glyparse::auto_parse()</code>. - A GGM database motif name (use <code>db_motif_info()</code> > <code>dplyr::filter(source_id == "GGM")</code> to inspect resolvable motif names).
...	These dots must be empty and are used only to force optional arguments to be supplied by name.
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 GGM 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 <code>have_motif()</code> , <code>count_motif()</code> , and <code>match_motif()</code> , this must be a logical vector with length 1 or the number of motif nodes (length 1 is recycled). For <code>have_motifs()</code> , <code>count_motifs()</code> , and <code>match_motifs()</code> , this must be a list of logical vectors with length equal to motifs; each element follows the same length rules. When <code>match_degree</code> 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-")
view_motif(glycan, motif)
```

Index

`.g_count_motif(dot-g_motif)`, 9
`.g_have_motif(dot-g_motif)`, 9
`.g_match_motif(dot-g_motif)`, 9

`branch_motifs`, 2
`branch_motifs()`, 11, 12

`count_motif`, 3
`count_motif()`, 6, 9, 10, 18, 20, 22, 25
`count_motifs(count_motif)`, 3
`count_motifs()`, 2, 6, 7, 11, 18, 20

`db_motif_info`, 8
`db_motif_info()`, 7, 14, 19, 21
`db_motifs`, 7
`db_motifs()`, 19
`dot-g_motif`, 9
`dynamic_motifs`, 11
`dynamic_motifs()`, 3, 13

`extract_branch_motif`, 12
`extract_branch_motif()`, 2, 3
`extract_motif`, 13
`extract_motif()`, 11

`get_motif_alignment`
 (`get_motif_structure`), 14
`get_motif_structure`, 14
`glydraw::draw_cartoon()`, 26
`glyparse::auto_parse()`, 4, 5, 12, 13, 15,
 16, 20, 23, 24, 26
`glyrepr::as_glycan_structure()`, 10
`glyrepr::get_structure_graphs()`, 9
`glyrepr::glycan_structure()`, 4, 5, 12–16,
 22–24, 26
`glyrepr::localize_floating_parts()`, 6,
 18, 25
`glyrepr::possible_linkages()`, 17

`have_motif`, 15
`have_motif()`, 3, 5, 6, 9, 10, 22, 25

`have_motifs(have_motif)`, 15
`have_motifs()`, 2, 6, 7, 11

`is_known_motif`, 21

`match_motif`, 22
`match_motif()`, 6, 9, 10, 18, 26
`match_motifs(match_motif)`, 22
`match_motifs()`, 2, 6, 11, 18

`view_motif`, 25