SugarPy

Release 1.0.0

Jan 08, 2021

Contents

The latest Documentation was generated on: Jan 08, 2021

CHAPTER 1

SugarPy

Universal, discovery-driven analysis of intact glycopeptides

1.1 Summary

SugarPy facilitates the glycan database independent, discovery-driven identification of intact glycopeptides from in-source collision-induced dissociation mass spectrometry data.

1.2 Abstract

Protein glycosylation is a complex post-translational modification with crucial cellular functions in all domains of life. Currently, large-scale glycoproteomics approaches rely on glycan database dependent algorithms and are thus unsuitable for discovery-driven analyses of glycoproteomes. Therefore, we devised SugarPy, a glycan database independent Python module, and validated it on the glycoproteome of human breast milk. We further demonstrated its applicability by analyzing glycoproteomes with uncommon glycans stemming from the green algae Chlamydomonas reinhardtii and the archaeon Haloferax volcanii. SugarPy also facilitated the novel characterization of glycoproteins from the red alga Cyanidioschyzon merolae.

1.3 Download and Installation

Get the latest version via GitHub: https://github.com/SugarPy/SugarPy

as a .zip package: https://github.com/SugarPy/SugarPy/archive/master.zip

or via git clone:

```
git clone https://github.com/SugarPy/SugarPy.git
```

Navidate into the downloaded/cloned folder and install the requirements:

```
user@localhost:~$ cd SugarPy
user@localhost:~/SugarPy$ pip install -r requirements.txt
```

After that, SugarPy can be installed into the Python sitepackages using:

```
user@localhost:~/SugarPy$ python setup.py install
```

Note: You might need administrator privileges to write in the Python site-package folder. On Linux or OS X, use `` `sudo python setup.py install` `` or write into a user folder by using this command `` `python setup.py install --user` ``. On Windows, you have to start the command line with administrator privileges.

1.4 Usage

SugarPy can be used as a standalone engine and some example scripts are provided in the example_scripts folder. However, we recommend to use it within the Python framework Ursgal , which combines various proteomics tools and therefore allows to easily set up a complete workflow. Example scripts for Ursgal workflows that include SugarPy are given in Ursgals example_scripts folder.

1.5 Documentation

For a more detailed documentation of SugarPy's structure, please refer to the documentation folder or https://sugarpy-ms.readthedocs.io

1.6 Questions and Participation

If you encounter any problems you can open up issues at GitHub, or write an email to sugarpy.team@gmail.com.

For any contributions, fork us at https://github.com/SugarPy/SugarPy and open up pull requests!

1.7 Disclaimer

While automatic filtering of SugarPy results leads to the reliable identification of glycopeptides in the cases we have tested, we recommend manual inspection of results and to alter filter criteria as needed.

1.8 Copyrights

- Anne Oltmanns
- Mecky Pohlschröder
- Stefan Schulze

1.9 Citation

Schulze, S.; Oltmanns, A.; Fufezan, C.; Krägenbring, J.; Mormann, M.; Pohlschröder, M.; Hippler, M. (2020). SugarPy facilitates the universal, discovery-driven analysis of intact glycopeptides. Bioinformatics

CHAPTER 2

Module structure

2.1 General Structure

SugarPy is comprised by a **run** and a **results** class. The run class includes all major functions to identify glycopeptides from IS-CID mass spectrometry measurements, while the results class can be used to generate different output files.

2.1.1 Run

The main functionalities of the SugarPy *run* class include

1. parsing protein database search result files to extract peptide sequences, retention times, etc.
2. building all theoretical combinations for a set of peptide sequences, monosaccharides and a given glycan length
3. searching MS1 spectra for isotope envelopes of all theoretical glycopeptides (using pyQms)
4. reassambling intact glycopeptides by combining subtrees, calculating SugarPy scores, etc.

2.1.2 Results

A variety of output files can be generated using the SugarPy *results* class, including

1. output .csv file
2. glycopeptide elution profiles
3. annotated spectra

Furthermore, the results class can be used to validate identified glycopeptides through non-IS-fragmenting MS runs. The corresponding .mzML files can be searched for glycopeptides on different levels

1. isotope envelope on MS1 level
2. selection for fragmentation by HCD
3. presence of glycopeptide specific fragment ions on MS2 level

CHAPTER 3

SugarPy module contents

3.1 SugarPy Run

class sugarpy.run.**Run**(*monosaccharides={}*)

The SugarPy run class comprises the main functionality for glycopeptide matching. A typical workflow contain the following steps: 1. parse_ident_file:

Ursgal result files are parsed and peptide sequences, as well as their modifications (except monosaccharides that would be part of the glycan) and retention times (RTs), are extracted.

2. **build_combinations and add_glycans2peptide:** For a set of monosaccharides (param: monosaccharides) and maximal glycan length (param: max_tree_length), all possible combinations of monosaccharides are calculated and the chemical compositions of the resulting theoretical glycans (taking into account glycans with the same mass) are added to the chemical compositions of the extracted set of peptides.
3. **quantify:** pyQms is used to build isotope envelope libraries for the theoretical glycopeptides and to match them against all MS1 spectra within the given RT windows. It should be noted that isotope envelopes consist of the theoretical m/z and relative intensity for all isotopic peaks. Therefore, the quality of the resulting matches is indicated by an mScore, which comprises the accuracies of the measured m/z and intensity.
4. **sort_results and validate_results:** For each matched molecule, a score (VL) is calculated as the length of a vector for the mScore (ranging from 0 to 1) and intensity (normalized by the maximum intensity of matched glycopeptides within the run, therefore also ranging from 0 to 1). For each spectrum, all matched molecules are sorted by the glycan length (number of monosaccharides). Subsequently, starting with the longest glycan, for each glycan length, all glycan compositions are checked if they are part of any glycan composition of the previous level (longer glycan). Glycan compositions that are true subsets of larger, matched glycans (subtrees of those) are considered fragment ions and are therefore merged with the larger, final glycopeptides. It should be noted that glycan compositions can be subtrees of multiple final glycans. Furthermore, fragmentation pathways are not taken into account, however, if Y1-ions are matched (peptide harboring one monosaccharide), the corresponding monosaccharide is noted as the reducing end. For all final glycopeptides within one spectrum, the subtree coverage is calculated. Finally, the SugarPy score is calculated for each glycopeptide as the

sum of vector lengths from all corresponding subtrees (fragment ions Y0 to Yn) multiplied by the subtree coverage.

add_glycans2peptide(*peptide_list=[]*, *max_tree_length=None*, *monosaccharides=None*)
Adds chemical composition of glycans to a given list of peptides. Peptides need to be in unimod style (Peptide#Modifications). The chemical composition of the original peptidoform is returned as well.

Keyword Arguments: peptide_list (list): List of peptides in unimod style max_tree_length (int): maximum number of monosaccharides in one combination monosaccharides(dict): dictionary containing name and chemical composition of monosaccharides

Returns: dict: { 'Sequence#Modifications : {glycan_hill_notation': ['Name']}}

build_combinations(*max_tree_length=None*, *monosaccharides=None*, *mode='replacement'*)
Builds and returns a dictionary containing chemical compositions of all combinations (with replacement, not ordered) of a given dict of monosaccharides and a maximal length of the tree.

Keyword arguments: max_tree_length (int): Maximum number of monosaccharides in one combination monosaccharides(dict): Dictionary containing name and chemical composition of monosaccharides

Returns:

dict: keys: chemical compositions of all combinations (with replacement, not ordered), values: combination(s) monosaccharide names corresponding to the chemical composition

ToDo: change monosaccharides to list and get compositions from ursgal.ChemicalComposition(), keyword argument for calculate_formula?

build_match_dict(*spectrum_dict=None*)
Uses a spec_collector (see sort_results) spectrum to extract the peptide and glycan composition and to sort glycans according to their length.

Keyword Arguments:

spectrum_dict (dict): dictionary for a spectrum from spec_collector (see sort_results)

Returns:

dict: { n (tree length) [[{] formula: , glycan_comp: , vector_length: , }, . . .]

}

build_rt_lookup(*mzml_file*, *ms_level*)
Builds and returns a dictionary equivalent to the ursgal_lookup.pkl It contains a dictionary (key is scan number, value is rt) for every mzml file name.

Arguments: mzML_file: Path to the mzML file. ms_level: MS level for which the lookup should be built

Returns: dict

extract_pep_and_glycan_comp(*trivial_name=None*)
Use the trivial name to extract information about the peptide and glycan composition. This also works for just extracting the glycan composition from a glycan string.

Keyword Argumens:

trivial_name('str'): trivial name of the glycan ('HexNAc(2)Hex(5)') or glycopeptide ('PEPTIDE|HexNAc(2)Hex(5)')

Returns: peptide(str): peptide sequence glycan_comp(dict): glycan composition as dictionary with monosaccharides as keys

and their number as value

parse_ident_file(*ident_file=None*, *unimod_glycans_incl_in_search=[]*)

Parses an Ursgal results .csv file and extracts identified peptides together with their retention times. Glycans that were included in the search as modifications are removed.

Keyword Arguments:

ident_file (str): Path to the Ursgal result .csv file. This file should only include (potential) glycopeptides, i.e. it should be filtered.

unimod_glycans_incl_in_search (list): List of Unimod PSI-MS names corresponding to glycans that were included in the database search as modification (will be removed from the peptide).

Returns:

dict: Lookup containing retention times and accuracies of all PSMs for each identified peptidoform (Peptide#Unimod:Pos)

quantify(*molecule_name_dict=None*, *rt_window=None*, *ms_level=1*, *charges=None*, *params=None*, *pkl_name=''*, *mzml_file=None*, *spectra=None*, *return_all=False*, *collect_precursor=False*, *force=False*)

Quantify a list of molecules in a given mzML file using pyQms. Quantification is done by default on MS1 level and can be specified for a retention time window.

Keyword Arguments:

molecule_name_dict (dict): contains for the molecules that should be quantified as hill notations (keys) a list of corresponding trivial names (values)

rt_window (dict): optional argument to define a retention time window in which the molecules are quantified (use 'min' and 'max' as keys in the dict)

ms_level: MS level for which quantification should be performed charges (list): list of charge states that are quantified params (dict): pyQms parameters (see pyQms manual for further information) pkl_name (str): name of the result pickle containing the pyQms results mzml_file (str): path to the mzML file used for the quantification spectra (list): optional list of spectrum IDs that should be quantified return_all (bool): if True, in addition to the results pkl, the IsotopologueLibrary

as well as the spectrum peaks are returned. This should only be used for a single spectrum.

Returns: str: path to the results pickle

sort_results(*results=None*, *min_spec_number=1*)

Parse through pyQms results, determines vector length for each matched spectrum with vectors beeing defined by the mScore and the normalized intensity (normalized scaling factor).

Keyword Arguments: min_spec_number (int): defines the minimum number of spectra for one matched formula. results (dict): pyQms results dictionary

Returns

dict: spec_collector = { matched_spectrum [{ formula][{] 'vector' : [], 'charge' : [], 'trivial_name' : [], 'glycan_comp' : [], 'glycan_trees' : [],

}

validate_results(*pyqms_results_dict=None*, *min_spec_number=0*, *min_tree_length=0*, *monosaccharides=None*)

Parse through pyQms results list and validate the results which includes the following: * sort_results: determines vector length for each matched spectrum with vectors beeing defined

by the mScore and the normalized intensity (normalized scaling factor). Also filters for a minimum number of spectra (for each molecule) in the results

- build_match_dict: extracts information about glycan compositions, sorts glycans by their length

- **starting with the longest glycans, for each level (glycan length) the corresponding glycans** are determined (glycans that are subtrees of longer glycans) and merged
- **the quality of glycan assignments is assessed by calculating the SugarPy_score ((Sum of vector lengths)*subtree** the subtree_coverage (Number of unique matched subtree lengths/Total number of unique subtree lengths) and the number of matched subtrees

Keyword Arguments: pyqms_results_dict (dict): dictionary containing the Peptides#Unimod (key) and corresponding pyqms result pkl (value) min_spec_number (int): defines the minimum number of spectra for one matched formula. min_tree_length (int): minimum number of monosaccharides per glycan monosaccharides (dict): dictionary containing name and chemical composition of monosaccharides

Returns

results class object (dict): class (dict) containing all scored_glycans as well as the spec_collector for every peptide_unimod

3.2 SugarPy Results

class sugarpy.results.**Results**(*scan_rt_lookup={}*, *validated_results=None*, *monosaccharides={}*)

SugarPy results are stored as a SugarPy results class in the Python pickle format. Employing functions from the results class allows to e.g.:

- write results as CSV files (write_results2csv)
- plot elution profiles (plot_glycan_elution_profile)
- plot annotated spectra (plot_annotated_spectra)

The SugarPy results class itself is a dictionary that contains all scored_glycans as well as the spec_collector for each peptide: dict = {

peptide_unimod [{]

'scored_glycans': {

spec1 [{]

glycan_tuple1 [{] 'tree_length': int 'SugarPy_score': float, 'num_subtrees': int, 'suc0r': set, 'formula': str, 'subtrees': list,

}, glycan_tuple1 : ...

}, spec2 : ...

}, 'spec_collector': {

spec1 [{]

formula1 [{] 'vector' : list, 'charge' : list, 'trivial_name' : list, 'glycan_comp' : list, 'glycan_trees' : list,

}, formula2 : ...

}, spec2: ...

}

}

}

add_results(*peptide_unimod=None*, *spec_collector=None*, *scored_glycans=None*)
Adds results to the SugarPy results class

Keyword Arguments: peptide_unimod (str): peptide#unimod spec_collector (dict):

dictionary returned by run.sort_results()

scored_glycans (dict): dictionary generated by run.validate_results()

calc_Y_ions(*glycan_combinations=None*, *peptide_unimod=None*, *charge=2*, *end_monosacch='HexNAx'*, *internal_precision=None*)

Returns: dict: { transformed mz: [name1, name2, . . .] }

calc_and_match_frag_ions(*glycan_list=[]*, *peptide_unimod=None*, *spec_id_list=[]*, *pymzml_run=None*, *internal_precision=None*)

Returns: dict: {spec_id: {'oxonium_ions' : [], 'Y_ions' : [],}}

calc_oxonium_ions(*glycan_combinations=None*, *internal_precision=None*)

Returns: dict: { transformed mz for z=1 : [name1, name2, . . .] }

check_peak_presence(*mzml_file=None, sp_result_file=None, ms_level=1, output_file='', pyqms_params=None, rt_border_tolerance=None, min_spec_number=1, charges=[1, 2, 3, 4, 5]*)
Takes a SugarPy result file as well as an mzML file to check in the mzML file for the presence of peaks corresponding to identified glycopeptides. If any are found, it is also checked if they were fragmented at some point of the run.

extract_best_matches(*sp_result_file=None*, *output_file='extracted_results.csv'*, *max_trees_per_spec=1*, *min_spec_number=1*)
Filter a SugarPy results csv file to extract the best matching glycan compositions.

Keyword Arguments: sp_result_file (str): inpput file path output_file: output file name max_trees_per_spec:

Maximum number of glycan compositions taken into account per spectrum

min_spec_number: Minimum number of consecutive spectra required for glycan to be accepted

glycan_to_tuple(*glycan*)
Converts a glycan (unimod style: Hex(2)HexNAc(5)) into a tuple of (monosaccharide, count) pairs

parse_result_file(*result_file*, *return_type='plot'*, *min_spec_number=1*)
Parses a SugarPy results .csv file and extracts identified peptides together with their glycans and charges.

Arguments: result_file (str): Path to the SugarPy result .csv file. return_type (str): 'plot' or 'peak_presence'

Returns:

dict: The dict contains all identified peptidoforms (Peptide#Unimod:Pos), as keys and a dict with the glycans (keys) and {'charges':set(), 'file_names':set()} (value) as values

plot_annotated_spectra(*mzml_file=None, plot_peak_types=['matched', 'unmatched', 'labels'], remove_subtrees=[], plot_molecule_dict=None, peak_colors={'labels': (0, 0, 200), 'matched': (0, 200, 0), 'raw': (100, 100, 100), 'unmatched': (200, 0, 0)}, ms_level=1, output_folder='', ms_precision='5e-6', plotly_layout=None*)
Plot one or multiple spectra (raw data). The following peaks can be added:

- matched (peaks matched by pyQms) and/or

- unmatched (unmatched peaks from matched formulas) peaks
- labels (for monoisotopic peaks)

plot_glycan_elution_profile(*peptide_list=None*, *min_sugarpy_score=0*, *min_sub_cov=0.0*, *x_axis_type='retention_time'*, *score_type='top_score'*, *output_file=None*, *title=None*, *scan_rt_lookup=None*, *plotly_layout=None*)

Plot elution profile(s) for identified glycopeptide(s)

Keyword Arguments: peptide_list (list): list of peptide#unimod for which elution profiles should be plotted output_file (str): output file name min_sugarpy_score (float):

minimum SugarPy score (glycan compositions with lower scores are not returned)

min_sub_cov (float): minimum subtree coverage (glycan compositions with sub_cov are not returned)

x_axis_type (str): Plot by spectrum_id or retention_time (x-axis)

score_type (dict): Plot by best score (top_score) or sum of scores (sum_scores) for each spectrum (y-axis)

title (str): Title of plot plotly_layout (dict): plotly layout used for the plot

Returns: str: output file name

plot_molecule_elution_profile(*plot_molecule_dict=None*, *output_file=None*, *title=None*, *include_subtrees='no_subtrees'*, *monosaccharides=None*, *scan_rt_lookup=None*, *x_axis_type='retention_time'*, *plotly_layout=None*)

Plot elution profile for molecules (chemical compositions). This can be used e.g. to plot separate elution profiles for fragmetn ions of a glycan composition

plot_scatter_vec_id(*x_axis_list=None*, *y_axis_list=None*, *text_list=None*, *name_list=None*, *title=None*, *x_axis_name=None*, *y_axis_name=None*, *output_file=None*, *plotly_layout=None*)

Plots a Scatter plot with given x and y data. Both need to be given as a list of lists, each list representing one trace in the final plot. Values can be annotated using a text_list. Traces can be annotated using a name_list.

sort_glycan_trees(*scored_glycan_trees=None*, *tuple_pos=1*, *sort_by='SugarPy_score'*)

Sort glycan composition e.g. by the SugarPy_score.

Returns: dict

sort_plot_lists(*name_list=None*, *x_axis_list=None*, *y_axis_list=None*, *text_list=None*)

Sort name_list alphabetically in a new list and append elements from x_axis_list, y_axis_list and text_list to new sorted lists in the right order.

write_results2csv(*output_file=None*, *max_trees_per_spec=5*, *min_sugarpy_score=0*, *min_sub_cov=0.0*, *peptide_lookup=None*, *monosaccharides=None*, *scan_rt_lookup=None*, *mzml_basename=None*)

Return a csv file containing a summary of all results stored in the results class

Keyword Arguments: output_file (str): output file name max_trees_per_spec (int):

maximum number of glycan compositions returned for one spectrum

min_sugarpy_score (float): minimum SugarPy score (glycan compositions with lower scores are not returned)

min_sub_cov (float): minimum subtree coverage (glycan compositions with sub_cov are not returned)

peptide_lookup (dict): dictionary returned by run.parse_ident_file()

monosaccharides (dict): dictionary containing name and chemical composition of monosaccharides

scan_rt_lookup (dict): dictionary containing the retention time for each spectrum

mzml_basename (str): name of the mzML or sample

Returns: str: output file name

3.3 Parameters

3.3.1 SugarPy Parameters

SugarPy is devided into a **run** and a **results** class. Some parameters are shared by both, while others are specific to the respective class.

General Parameters

- **charges (list):** list of charge states that are quantified
- **max_tree_length (int):** maximum number of monosaccharides in one combination
- **min_spec_number (int):** defines the minimum number of spectra for one matched formula.
- **monosaccharides(dict):** dictionary containing name and chemical composition of monosaccharides
- **ms_level:** MS level for which quantification should be performed
- **mzml_file (str):** path to the mzML file used for the quantification
-

Run Parameters

For an example on how to use these parameters, refer to sugarpy_run_example.py

- **ident_file (str):** Path to the Ursgal result .csv file. This file should only include (potential) glycopeptides, i.e. it should be filtered.
- **min_tree_length (int):** minimum number of monosaccharides per glycan
- **params (dict):** pyQms parameters (see pyQms manual for further information)
- **pkl_name (str):** name of the result pickle containing the pyQms results
- **rt_window (dict):** optional argument to define a retention time window in which the molecules are quantified (use 'min' and 'max' as keys in the dict)
- **spectra (list):** optional list of spectrum IDs that should be quantified
- **unimod_glycans_incl_in_search (list):** List of Unimod PSI-MS names corresponding to glycans that were included in the database search as modification (will be removed from the peptide).

Results Parameters

For an example on how to use these parameters, refer to sugarpy_results_example.py

- **output_file (str):** output file name
- **max_trees_per_spec (int):** maximum number of glycan compositions returned for one spectrum
- **min_sugarpy_score (float):** minimum SugarPy score (glycan compositions with lower scores are not returned)
- **min_sub_cov (float):** minimum subtree coverage (glycan compositions with sub_cov are not returned)

CHAPTER 4

Examples

Different example script are given to test SugarPy's functionality and can be used as templates for your own script. For example scripts on how to use SugarPy withing the Python framework Ursgal, please see https://ursgal.readthedocs.io/en/latest/example_scripts.html

4.1 Example Scripts

This collection of example scripts is designed to get familiar with the basic functionality of SugarPy. They can be modified and combined according to your needs. Please also check the folder "example_scripts" as well as Ursgal's "example_scripts" for further use cases and implementation into more compelx workflows.

4.1.1 Simple Example Scripts

Parse peptide sequence input file

parse_peptide_sequence_input_file.**main**(*input_path=None*)

This script generates a peptide lookup that can be used as an input for further SugarPy functions. Here, an Ursgal result file is parsed and pepide sequences as well as corresponding informations are stored in a dictionary. However, This dictionary can also be generated in different ways, but should follow this structure:

```
peptide_lookup = {
    'sequence#unimod:pos': {
        'rt': set(),
        'spec_id': set(),
        'accuracy': [],
        'protein': 'description',
    }
}
```

Usage:

```
./parse_peptide_sequence_input_file.py <inpu_result_file.csv>
```

```
#!/usr/bin/env python3
# encoding: utf-8

import sugarpy
import sys
import pprint

def main(input_path=None):
    '''
    This script generates a peptide lookup that can be used as an input
    for further SugarPy functions. Here, an Ursgal result file is parsed
    and pepide sequences as well as corresponding informations are stored
    in a dictionary. However, This dictionary can also be generated in
    different ways, but should follow this structure::

        peptide_lookup = {
            'sequence#unimod:pos': {
                'rt': set(),
                'spec_id': set(),
                'accuracy': [],
                'protein': 'description',
            }
        }

    Usage::

        ./parse_peptide_sequence_input_file.py <inpu_result_file.csv>
    '''
    sp_run = sugarpy.run.Run()

    peptide_lookup = sp_run.parse_ident_file(
        ident_file=input_path,
        unimod_glycans_incl_in_search=[
            'HexNAc',
            'HexNAc(2)',
        ],
    )
    pprint.pprint(peptide_lookup)

if __name__ == '__main__':
    main(input_path=sys.argv[1])
```

Build glycopeptide combinations

build_glycopeptide_combinations.**main**()

For a given list of peptides and a set of monosaccharides, this script builds all theoretical combinations of glycopeptides.

Usage:

```
./build_glycopeptide_combination.py
```

```
#!/usr/bin/env python3
# encoding: utf-8

import sugarpy
import pprint

def main():
    '''
    For a given list of peptides and a set of monosaccharides,
    this script builds all theoretical combinations of glycopeptides.

    Usage::

        ./build_glycopeptide_combination.py
    '''
    sp_run = sugarpy.run.Run(
        monosaccharides={
            "dHex": 'C6H10O4',
            "Hex": 'C6H10O5',
            "HexNAc": 'C8H13NO5',
            "NeuAc": 'C11H17NO8',
        },
    )

    pep_unimod_list = [
        'GLYCANSTPEPTIDE',
    ]

    glycopeptide_combinations = sp_run.add_glycans2peptide(
        peptide_list=pep_unimod_list,
        max_tree_length=5,
    )

    pprint.pprint(glycopeptide_combinations)

if __name__ == '__main__':
    main()
```

4.1.2 Full Workflow Example Scripts

SugarPy run example

sugarpy_run_example.**main**(*ident_file=None, unimod_glycans_incl_in_search=['HexNAc', 'HexNAc(2)'], max_tree_length=10, monosaccharides={'Hex': 'C6H10O5', 'HexNAc': 'C8H13NO5', 'NeuAc': 'C11H17NO8', 'dHex': 'C6H10O4'}, mzml_file=None, scan_rt_lookup=None, rt_border_tolerance=1, charges=[1, 2], output_file=None, pyqms_params={}, min_spec_number=1, min_tree_length=1, max_trees_per_spec=5, min_sugarpy_score=0, min_sub_cov=0.5, ms_level=1, force=True*)

This example script takes the following two files as input and than performs a search for intact glycopeptides:

- Ursgal result file: this file contains the identified glycopeptide sequences in .csv format
- mzML file: the raw MS data in .mzML format

Usage:

```
./sugarpy_run_example.py <ursgal_result_file.csv> <ms_data_file.mzML>
```

```
#!/usr/bin/env python
# encoding: utf-8

import sugarpy
import ursgal
import sys
import os
import statistics
import pickle
from collections import namedtuple


def main(
    ident_file=None,
    unimod_glycans_incl_in_search=[
        'HexNAc',
        'HexNAc(2)',
    ],
    max_tree_length=10,
    monosaccharides={
        "dHex": 'C6H10O4',
        "Hex": 'C6H10O5',
        "HexNAc": 'C8H13NO5',
        "NeuAc": 'C11H17NO8',
    },
    mzml_file=None,
    scan_rt_lookup=None,
    rt_border_tolerance=1,
    charges=[1, 2],
    output_file=None,
    pyqms_params={},
    min_spec_number=1,
    min_tree_length=1,
    max_trees_per_spec=5,
    min_sugarpy_score=0,
    min_sub_cov=0.5,
    ms_level=1,
    force=True,
):
    '''
    This example script takes the following two files as input and
    than performs a search for intact glycopeptides:

    * Ursgal result file: this file contains the identified glycopeptide sequences in␣
↪.csv format
    * mzML file: the raw MS data in .mzML format

    Usage::

        ./sugarpy_run_example.py <ursgal_result_file.csv> <ms_data_file.mzML>
    '''

    # initialize Sugary run
    sp_run = sugarpy.run.Run(
        monosaccharides=monosaccharides
```

(continues on next page)

(continued from previous page)

```
    )

    # read the Ursgal result file and store pepides in lookup
    peptide_lookup = sp_run.parse_ident_file(
        ident_file=ident_file,
        unimod_glycans_incl_in_search=unimod_glycans_incl_in_search,
    )

    # build all theoretical glycopeptide combinations
    pep_unimod_list = list(peptide_lookup.keys())
    peps_with_glycans = sp_run.add_glycans2peptide(
        peptide_list=pep_unimod_list,
        max_tree_length=max_tree_length,
    )

    lookup_dict = sp_run.build_rt_lookup(mzml_file, ms_level)
    mzml_basename = os.path.basename(mzml_file).replace('.mzML', '')
    scan2rt = lookup_dict[mzml_basename]['scan_2_rt']
    output_folder = os.path.dirname(output_file)

    # run pyqms to identify isotope envelopes for all theoretical glycopeptides
    pyqms_results = {}
    for n, peptide_unimod in enumerate(sorted(peps_with_glycans)):
        pkl_name = '-'.join(peptide_unimod.split(';'))
        pkl_name = '_'.join(pkl_name.split(':'))
        rt_min = min(peptide_lookup[peptide_unimod]
                     ['rt']) - rt_border_tolerance
        rt_max = max(peptide_lookup[peptide_unimod]
                     ['rt']) + rt_border_tolerance
        print(
            '[ SugarPy  ] Quantification for peptide ',
            peptide_unimod,
            '#{0} out of {1}'.format(
                n + 1,
                len(peps_with_glycans)
            )
        )
        results_pkl = sp_run.quantify(
            molecule_name_dict=peps_with_glycans[peptide_unimod],
            rt_window=(rt_min, rt_max),
            ms_level=ms_level,
            charges=charges,
            params=pyqms_params,
            pkl_name=os.path.join(
                output_folder,
                '{0}_{1}_pyQms_results.pkl'.format(
                    mzml_basename,
                    pkl_name
                )
            ),
            mzml_file=mzml_file,
        )
        pyqms_results[peptide_unimod] = results_pkl

    # combine matched glycopeptide fragments to inteact glycopeptides
    validated_results = sp_run.validate_results(
        pyqms_results_dict=pyqms_results,
```

(continues on next page)

(continued from previous page)

```
        min_spec_number=min_spec_number,
        min_tree_length=min_tree_length,
    )

    # initiate results class and save Sugary results as csv file
    sp_results = sugarpy.results.Results(
        validated_results=validated_results,
        monosaccharides=monosaccharides
    )
    sp_results.write_results2csv(
        output_file=output_file,
        max_trees_per_spec=max_trees_per_spec,
        min_sugarpy_score=min_sugarpy_score,
        min_sub_cov=min_sub_cov,
        peptide_lookup=peptide_lookup,
        scan_rt_lookup=scan2rt,
        mzml_basename=mzml_basename,
    )
    output_pkl = output_file.replace('.csv', '.pkl')
    with open(output_pkl, 'wb') as out_pkl:
        pickle.dump(sp_results, out_pkl)
    print('Done.')

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print(main.__doc__)
        sys.exit(1)
    mzml_file = sys.argv[1]
    ident_file = sys.argv[2]
    output_file = ident_file.replace('.csv', '_sugarpy.csv')
    main(
        ident_file=ident_file,
        mzml_file=mzml_file,
        output_file=output_file,
    )
```

SugarPy results example

sugarpy_results_example.**main**(*mzml_file=None, validated_results_pkl=None, ursgal_result_file=None, ms_level=1, output_file=None, min_sugarpy_score=1, min_sub_cov=0.5, min_spec_number=1, charges=[1, 2, 3, 4, 5], monosaccharides={'Hex': 'C6H10O5', 'HexNAc': 'C8H13NO5', 'NeuAc': 'C11H17NO8', 'dHex': 'C6H10O4'}, rt_border_tolerance=1, max_tree_length=10, unimod_glycans_incl_in_search=['HexNAc', 'HexNAc(2)'], max_trees_per_spec=5*)

This example script takes the following three files as input:

- Ursgal result file: this file contains the identified glycopeptide sequences in .csv format
- mzML file: the raw MS data in .mzML format
- SugarPy results pkl: pkl file that contains SugarPy results (see sugarpy_run_exampe.py on how to generate it)

From those files, a SugarPy results csv is generated, as well as a glycopeptide elution profile.

Usage:

```
./sugarpy_results_example.py <ursgal_result_file.csv> <ms_data_file.mzML>
↪<sugarpy_results.pkl>
```

```
#!/usr/bin/env python
# encoding: utf-8

import sugarpy
import ursgal
import sys
import os
import pickle

def main(
    mzml_file=None,
    validated_results_pkl=None,
    ursgal_result_file=None,
    ms_level=1,
    output_file=None,
    min_sugarpy_score=1,
    min_sub_cov=0.5,
    min_spec_number=1,
    charges=[1, 2, 3, 4, 5],
    monosaccharides={
        "dHex": 'C6H10O4',
        "Hex": 'C6H10O5',
        "HexNAc": 'C8H13NO5',
        "NeuAc": 'C11H17NO8',
    },
    rt_border_tolerance=1,
    max_tree_length=10,
    unimod_glycans_incl_in_search=[
        'HexNAc',
        'HexNAc(2)',
    ],
    max_trees_per_spec=5,
):
    '''
    This example script takes the following three files as input:

    * Ursgal result file: this file contains the identified glycopeptide sequences in␣
↪.csv format
    * mzML file: the raw MS data in .mzML format
    * SugarPy results pkl: pkl file that contains SugarPy results (see sugarpy_run_
↪exampe.py on how to generate it)

    From those files, a SugarPy results csv is generated, as well as a glycopeptide␣
↪elution profile.

    Usage::

        ./sugarpy_results_example.py <ursgal_result_file.csv> <ms_data_file.mzML>
↪<sugarpy_results.pkl>
    '''

    # load results from pkl
    with open(validated_results_pkl, 'rb') as results_pkl:
```

(continues on next page)

(continued from previous page)

```
        validated_results = pickle.load(results_pkl)

    mzml_basename = os.path.basename(mzml_file)
    sp_run = sugarpy.run.Run()
    lookup_dict = sp_run.build_rt_lookup(mzml_file, ms_level)
    scan2rt = lookup_dict[mzml_basename]['scan_2_rt']

    # load results class
    sp_results = sugarpy.results.Results(
        monosaccharides=monosaccharides,
        scan_rt_lookup=scan2rt,
        validated_results=validated_results,
    )

    # write results csv
    sp_run = sugarpy.run.Run()
    peptide_lookup = sp_run.parse_ident_file(
        ident_file=ursgal_ident_file,
        unimod_glycans_incl_in_search=unimod_glycans_incl_in_search
    )
    sp_result_file = sp_results.write_results2csv(
        output_file=validated_results_pkl.replace('.pkl', '.csv'),
        max_trees_per_spec=max_trees_per_spec,
        min_sugarpy_score=min_sugarpy_score,
        min_sub_cov=min_sub_cov,
        peptide_lookup=peptide_lookup,
        monosaccharides=monosaccharides,
        scan_rt_lookup=scan2rt,
        mzml_basename=mzml_basename
    )


    # plot glycopeptide elution profile
    plot_molecule_dict = sp_results.parse_result_file(sp_result_file)
    elution_profile_file = sp_results.plot_glycan_elution_profile(
        peptide_list=sorted(plot_molecule_dict.keys()),
        min_sugarpy_score=min_sugarpy_score,
        min_sub_cov=min_sub_cov,
        x_axis_type='retention_time',
        score_type='top_scores',
        output_file=output_file.replace('.txt', '_glycan_profile.html'),
        plotly_layout={
            'font' : {
                'family':'Arial',
                'size': 26,
                'color' : 'rgb(0, 0, 0)',
            },
            'width':1700,
            'height':1200,
            'margin':{
                'l':120,
                'r':50,
                't':50,
                'b':170,
            },
            'xaxis':{
                'type':'linear',
```

(continues on next page)

(continued from previous page)

```
                'color':'rgb(0, 0, 0)',
                'title': 'Retention time [min]',
                'title_font':{
                    'family':'Arial',
                    'size':26,
                    'color':'rgb(0, 0, 0)',
                },
                'autorange':True,
                'fixedrange':False,
                'tickmode':'auto',
                'showticklabels':True,
                'tickfont':{
                    'family':'Arial',
                    'size':22,
                    'color':'rgb(0, 0, 0)',
                },
                'tickangle':0,
                'ticklen':5,
                'tickwidth':1,
                'tickcolor':'rgb(0, 0, 0)',
                'ticks':'outside',
                'showline':True,
                'linecolor':'rgb(0, 0, 0)',
                'mirror':False,
                'showgrid':False,
                'anchor':'y',
                'side':'bottom',
            },
            'yaxis':{
                'type':'linear',
                'color':'rgb(0, 0, 0)',
                'title':'Max. SugarPy score',
                'title_font':{
                    'family':'Arial',
                    'size':26,
                    'color':'rgb(0, 0, 0)',
                },
                'autorange':True,
                'fixedrange':False,
                'tickmode':'auto',
                'showticklabels':True,
                'tickfont':{
                    'family':'Arial',
                    'size':22,
                    'color':'rgb(0, 0, 0)',
                },
                'tickangle':0,
                'ticklen':5,
                'tickwidth':1,
                'tickcolor':'rgb(0, 0, 0)',
                'ticks':'outside',
                'showline':True,
                'linecolor':'rgb(0, 0, 0)',
                'mirror':False,
                'showgrid':False,
                'zeroline':False,
                'anchor':'x',
```

(continues on next page)

(continued from previous page)

```
                'side':'left',
            },
            'legend':{
                'font':{
                    'family':'Arial',
                    'size':5,
                    'color':'rgb(0, 0, 0)',
                },
                'orientation':'v',
                'traceorder':'normal',
            },
            'showlegend':True,
            'paper_bgcolor':'rgba(0,0,0,0)',
            'plot_bgcolor':'rgba(0,0,0,0)',
        },
    )

    print('Done.')
    return output_file


if __name__ == '__main__':
    ursgal_ident_file = sys.argv[1]
    mzml_file = sys.argv[2]
    validated_results_pkl = sys.argv[3]
    output_file = result_file.replace('.csv', '.html')
    main(
        mzml_file=mzml_file,
        validated_results_pkl=validated_results_pkl,
        ursgal_result_file=ursgal_result_file,
        output_file=output_file,
    )
```

CHAPTER 5

Changelog

Record of released SugarPy versions with all notable changes

5.1 Changelog

5.1.1 Version 1.0.0 (02.2019)

Initial version of SugarPy, as described in the corresponding publication.

Python Module Index

S

Index